Non-Cooperative Scheduling of Multiple Bag-of-Tasks Applications

TU Wien Seminar

Arnaud Legrand
(joint work with C. Touati and S. Hunold)

CNRS/INRIA/Grenoble University
LIG laboratory, MESCAL project

April 4, 2013

Motivation

Large-scale distributed platforms result from the collaboration of many users:

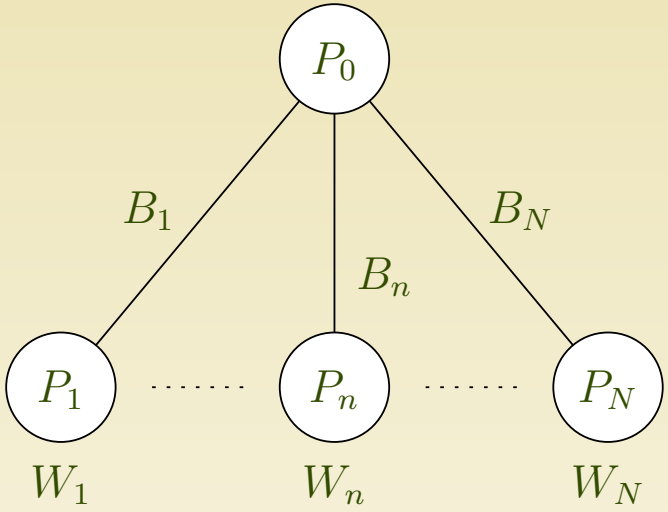
- Multiple applications execute concurrently on heterogeneous platforms and compete for CPU and network resources.
- Sharing resources amongst users should somehow be fair.
- Large scale platform require distributed scheduling.

In a grid or Volunteer Computing context, sharing is generally done in the "low" layers (network, OS).

1. We analyze the behavior of K non-cooperative schedulers that use the optimal strategy to maximize their own utility while fair sharing is ensured at a system level ignoring applications characteristics [INFOCOM'07].
2. We use Lagrangian optimization and distributed gradient descent to propose a fair and distributed scheduling algorithm for this framework [JPDC'13?].

Outline

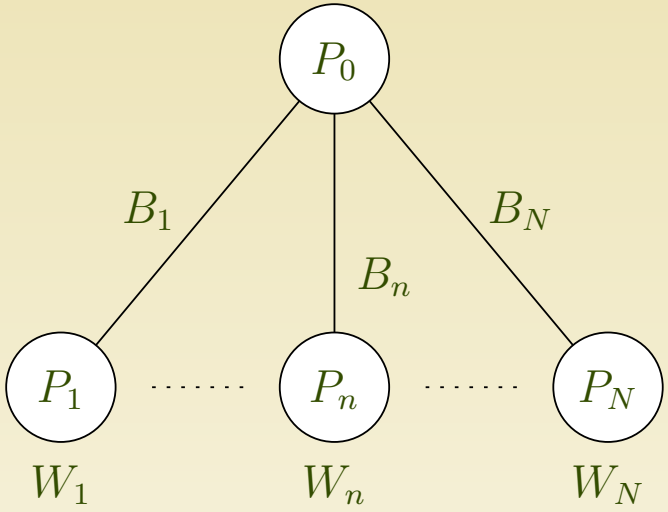
Master-Worker Platform



- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

Master-Worker Platform



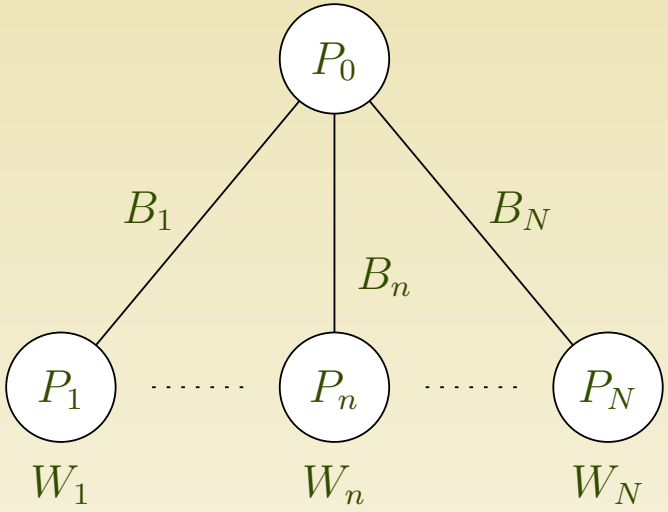
- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

- Multi-port

Communications to P_i do not interfere with communications to P_j.

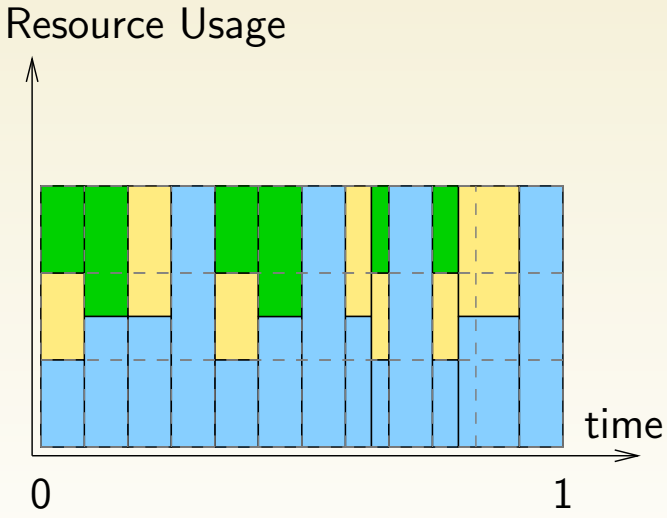
Master-Worker Platform



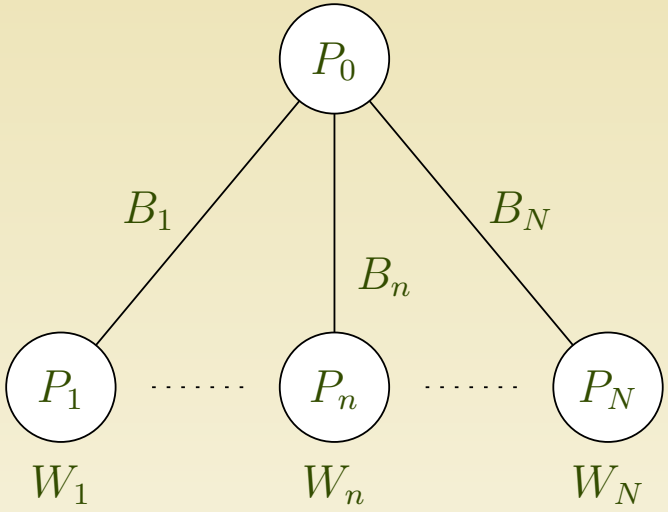
- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

- Multi-port
- No admission policy but an ideal local fair sharing of resources among the various requests



Master-Worker Platform



- N processors with processing capabilities W_n (in Mflop.s^{-1})
- using links with capacity B_n (in Mb.s^{-1})

Hypotheses :

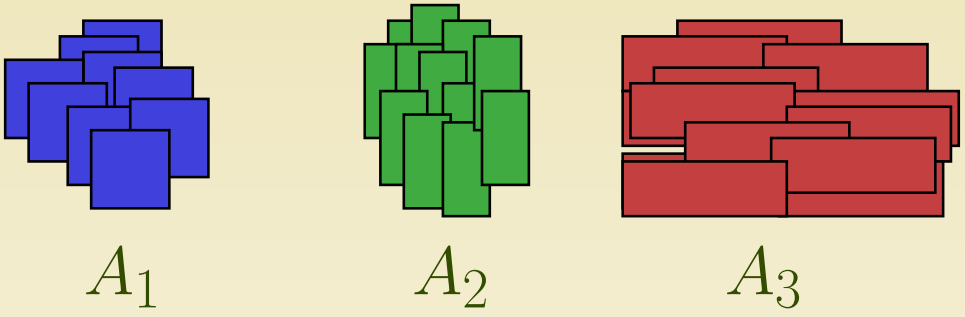
quests

Definition.

We denote by platform-system a triplet (N, B, W) where N is the number of machines, and B and W the vectors of size N containing the link capacities and the computational powers of the machines.

Applications

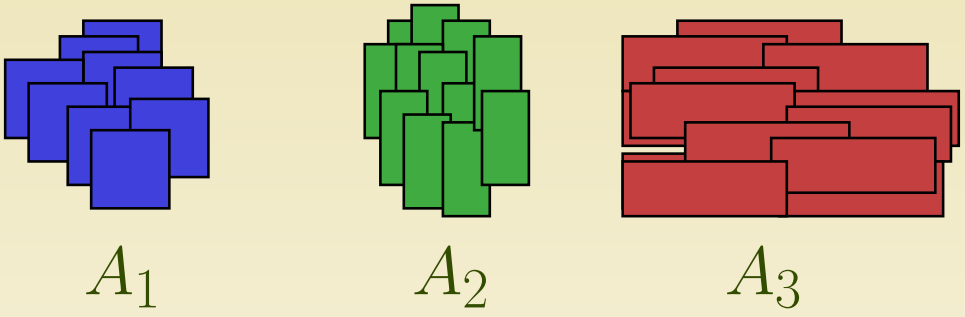
- Multiple applications ($A_1, \ldots, A_K$):



  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)

Applications

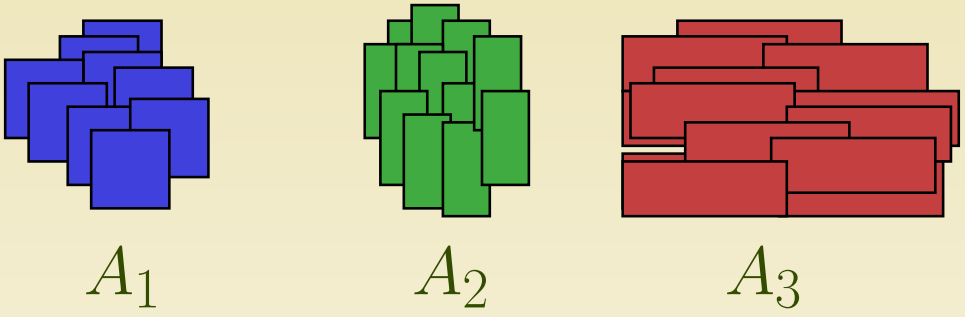
- Multiple applications ($A_1, \ldots, A_K$):



  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)
- Master holds all tasks initially, communication for input data only (no result message).

Applications

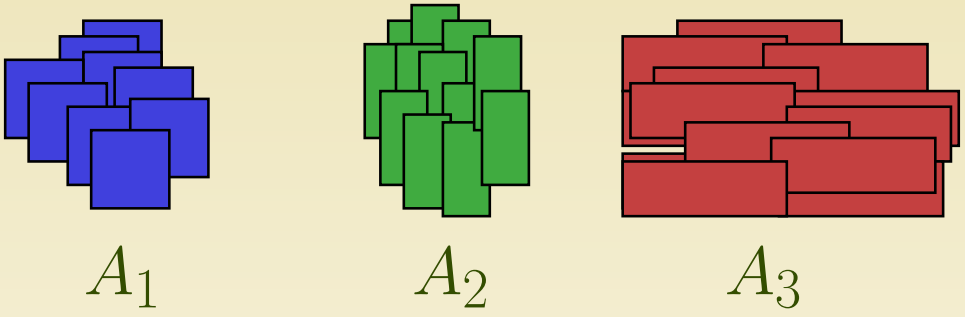
- Multiple applications ($A_1, \ldots, A_K$):



  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)
- Master holds all tasks initially, communication for input data only (no result message).
- Such applications are typical Desktop Grid or Volunteer Computing applications (SETI@home, Einstein@Home, ...)

Applications

- Multiple applications $(A_1, \ldots, A_K)$:



  - each consisting in a large number of same-size independent tasks
  - Different communication and computation demands for different applications. For each task of A_k:
    - processing cost w_k (MFlops)
    - communication cost b_k (MBytes)

Definition.

We define an application-system as a triplet (K, b, w) where K is the number of applications, and b and w the vectors of size K representing the size and the amount of computation associated to the different applications.

Steady-State scheduling

In the following our K applications run on our N workers and compete for network and CPU access:

Definition.

A system S is a sextuplet (K, b, w, N, B, W), with K,b,w,N,B,W defined as for a user-system and a platform-system.

- Task regularity $\rightsquigarrow$ steady-state scheduling.
- Maximize throughput (average number of tasks processed per unit of time)

$$\rho_k = \lim_{t \to \infty} \frac{done_k(t)}{t}.$$

  Similarly: $\rho_{n,k}$ is the average number of tasks of type k performed per time-unit on the processor P_n.

$$\rho_k = \sum_n \rho_{n,k}.$$

- ρ_k is the utility of application k.

Constraints

The scheduler of each application thus aims at maximizing its own throughput, i.e. ρ_k .

However, as applications use the same set of resources, we have the following general constraints:

Computation $\forall n \in [\![0, N]\!] : \sum_{k=1}^{K} \rho_{n,k} \cdot w_k \leqslant W_n$

Communication $\forall n \in [\![1, N]\!] : \sum_{k=1}^{K} \rho_{n,k} \cdot b_k \leqslant B_n$

Applications should decide:

- which worker to use,
- when to send data from the master to a worker,
- when to use a worker for computation.

Outline

Optimal strategy for a single application

Single application

This problem reduces to maximizing $\sum_{n=1}^{N} \rho_{n,1}$ while:

$$\begin{cases} \forall n \in [\![A, N]\!] : \rho_{n,1} \cdot w_1 \leqslant W_n \\ \forall n \in [\![1, N]\!] : \rho_{n,1} \cdot b_1 \leqslant B_n \\ \forall n, \quad \rho_{n,1} \geqslant 0. \end{cases}$$

The optimal solution to this linear program is obtained by setting

$$\forall n,\ \rho_{n,1} = \min\left(\frac{W_n}{w_1}, \frac{B_n}{b_1}\right)$$

In other words

The master process should saturate each worker by sending it as many tasks as possible.

A simple acknowledgment mechanism enables the master process to ensure that it is not over-flooding the workers, while always converging to the optimal throughput.

A Non-Cooperative Game

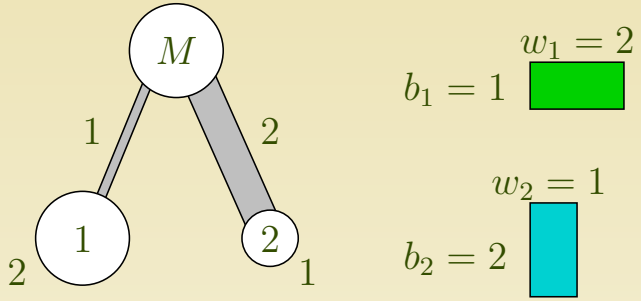
We suppose a purely non-cooperative game where no scheduler decides to "ally" to any other (i.e. no coalition is formed).

As the players constantly adapt to each others' actions, they may (or not) reach some equilibrium, known in game theory as Nash equilibrium [Nash51].

$\rho_{n,k}^{(nc)}$ denotes the rates achieved at such stable states (if any).

A simple example

Two computers 1 and 2: $B_1 = 1$, $W_1 = 2$, $B_2 = 2$, $W_2 = 1$.
Two applications: $b_1 = 1$, $w_1 = 2$, $b_2 = 2$ and $w_2 = 1$.

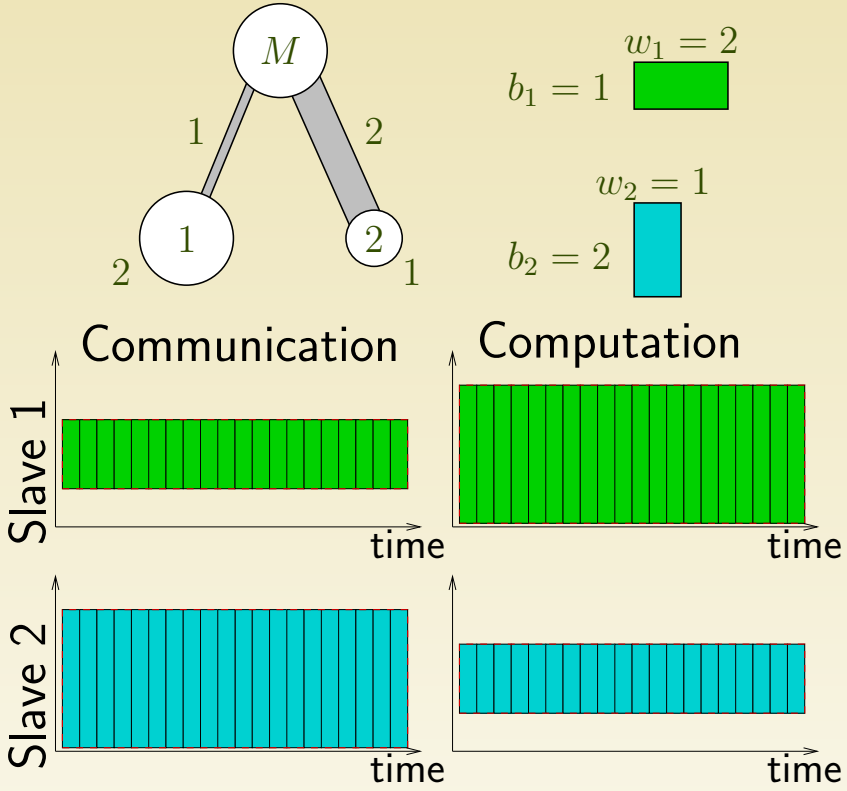


A simple example

Two computers 1 and 2: $B_1 = 1$, $W_1 = 2$, $B_2 = 2$, $W_2 = 1$.
Two applications: $b_1 = 1$, $w_1 = 2$, $b_2 = 2$ and $w_2 = 1$.

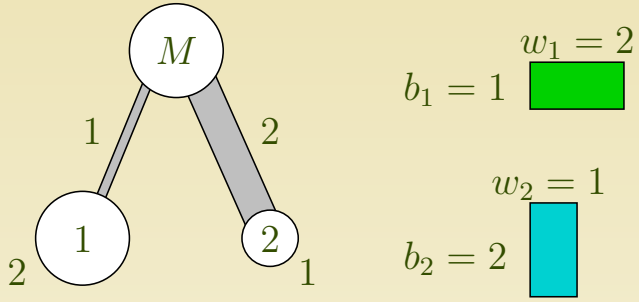
Cooperative Approach:

Application 1 is processed exclusively on computer 1 and application 2 on computer 2. The respective throughput is $\rho_1^{(\text{coop})} = \rho_2^{(\text{coop})} = 1$.



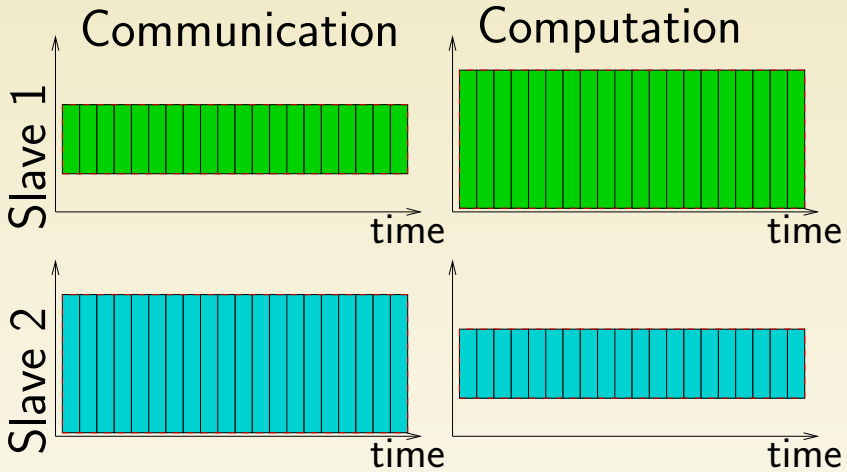
A simple example

Two computers 1 and 2: $B_1 = 1$, $W_1 = 2$, $B_2 = 2$, $W_2 = 1$.
Two applications: $b_1 = 1$, $w_1 = 2$, $b_2 = 2$ and $w_2 = 1$.



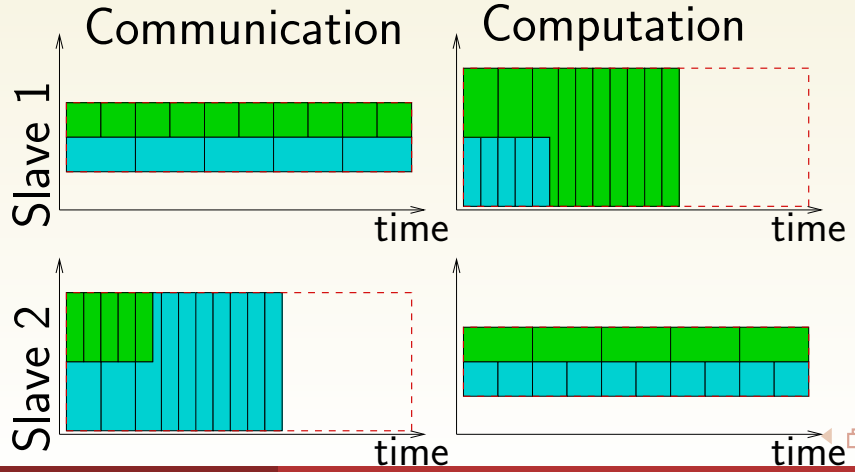
Cooperative Approach:

Application 1 is processed exclusively on computer 1 and application 2 on computer 2. The respective throughput is $\rho_1^{(\text{coop})} = \rho_2^{(\text{coop})} = 1$.

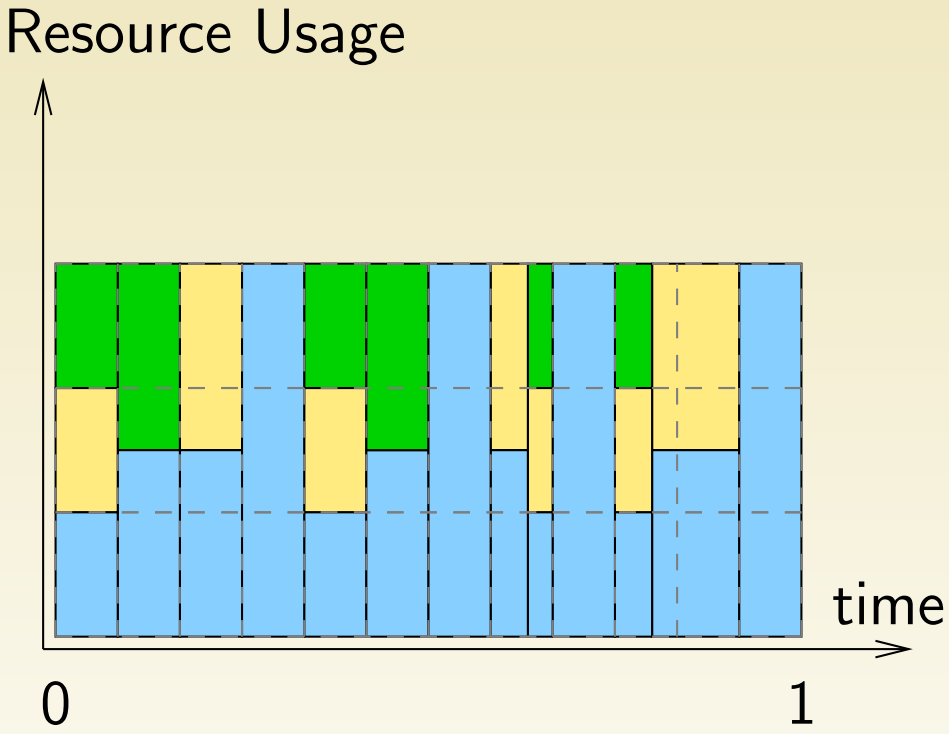


Non-Cooperative Approach:

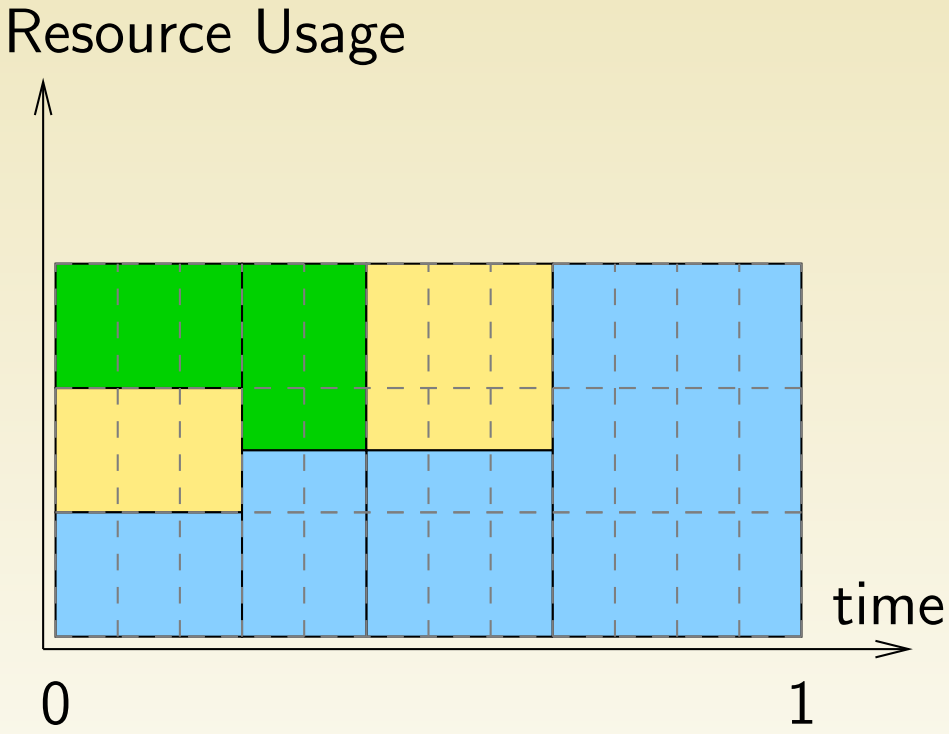
$\rho_1^{(nc)} = \rho_2^{(nc)} = \frac{3}{4}$



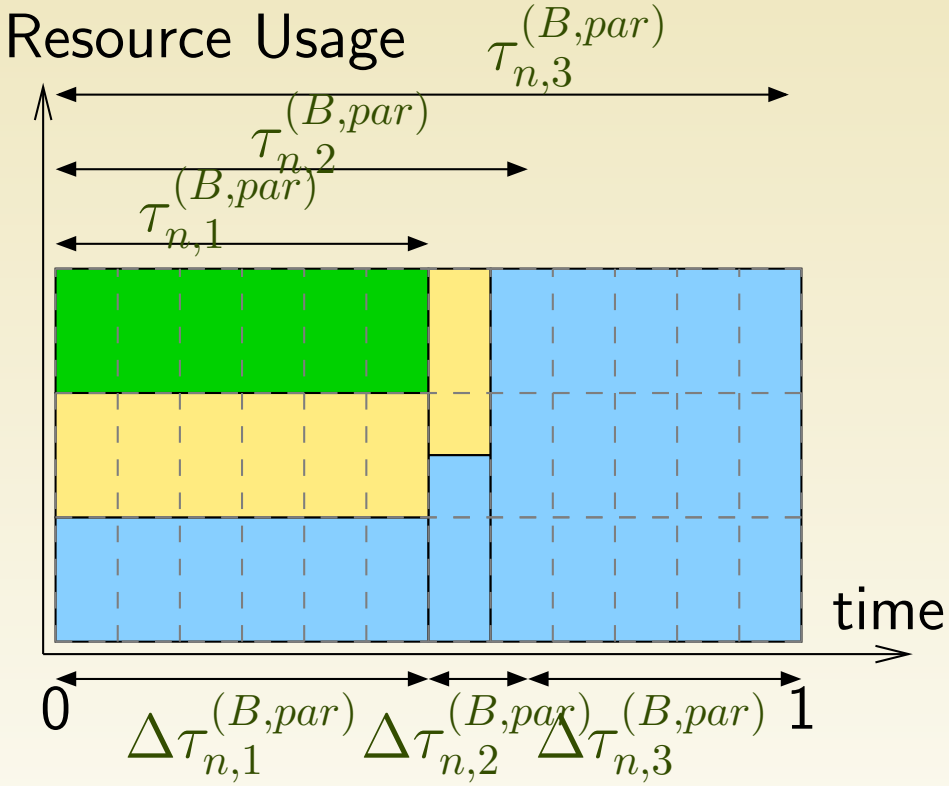
Characterizing the Nash Equilibrium: proof idea



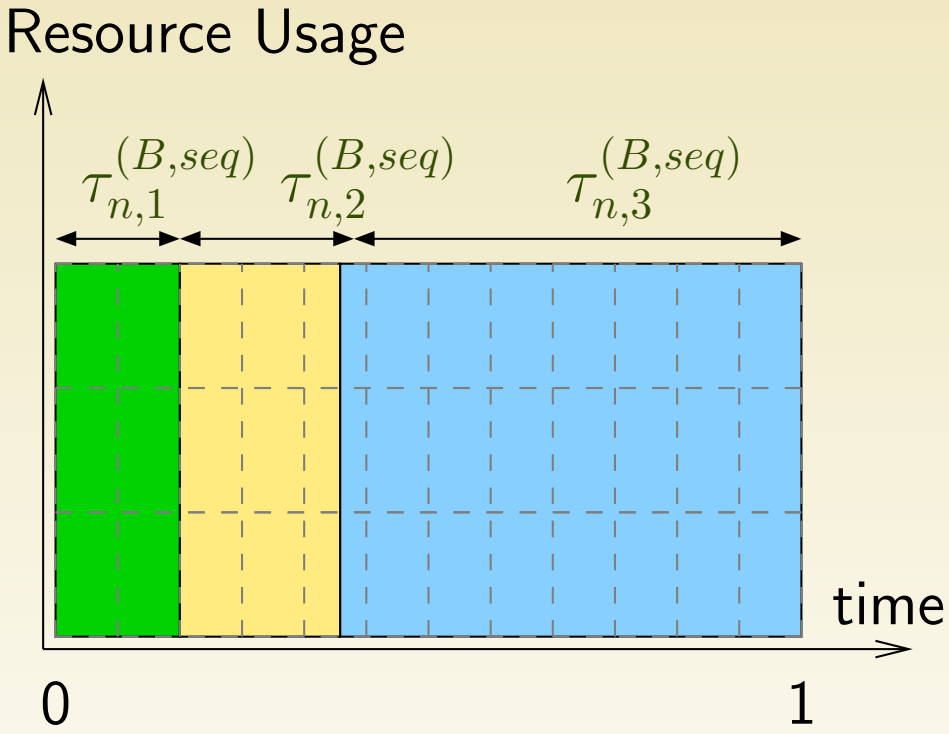
Characterizing the Nash Equilibrium: proof idea



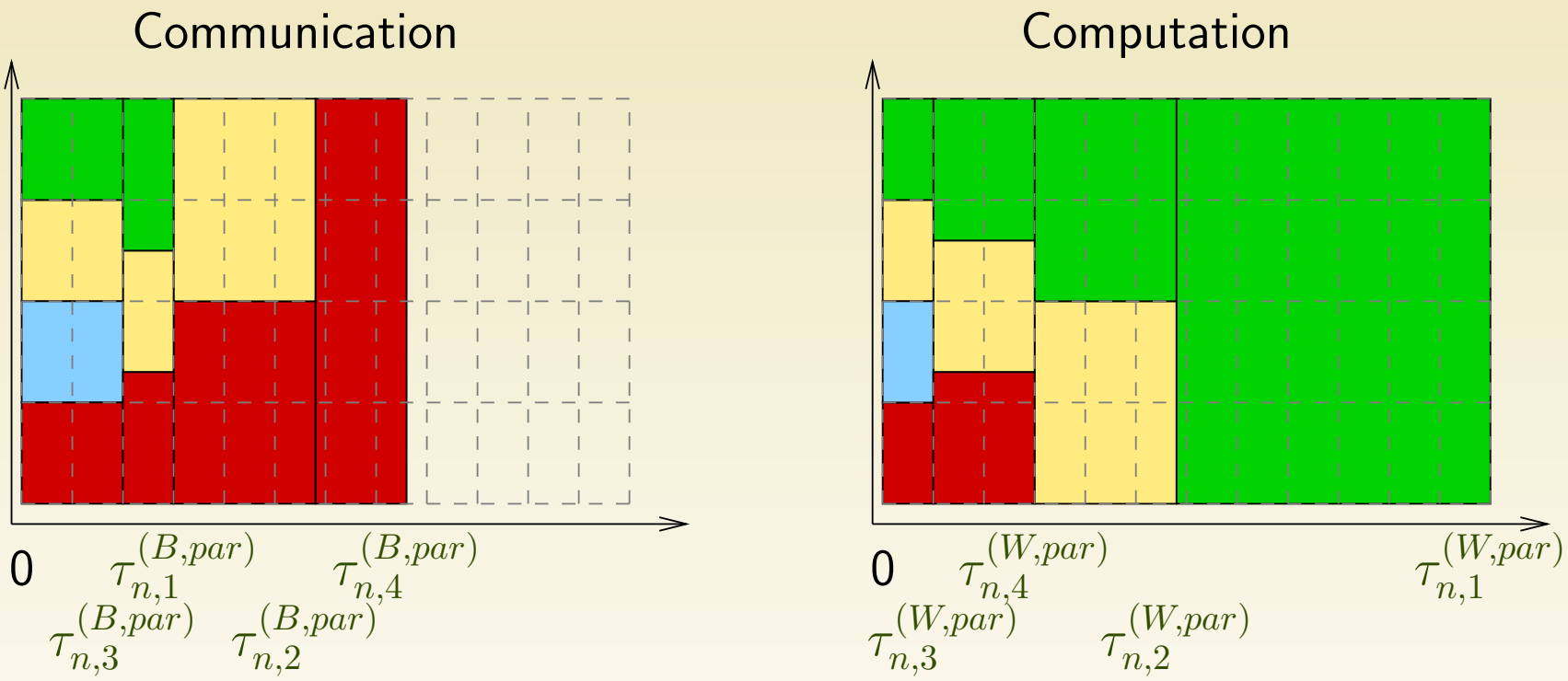
Characterizing the Nash Equilibrium: proof idea



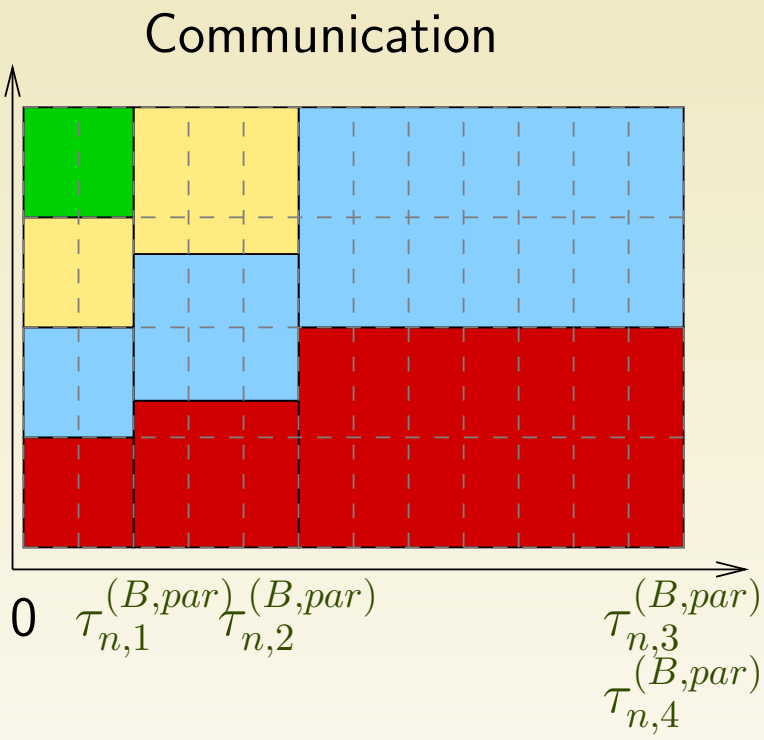
Characterizing the Nash Equilibrium: proof idea

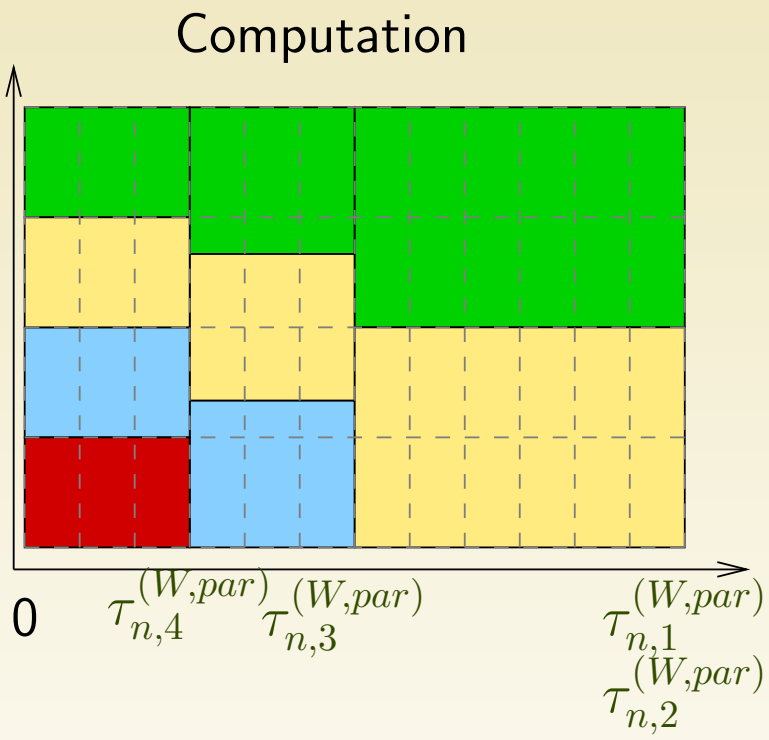


Characterizing the Nash Equilibrium: proof idea



Characterizing the Nash Equilibrium: proof idea





Characterizing the Nash Equilibrium: proof idea

We assume $c_1 \leqslant c_2 \leqslant \cdots \leqslant c_K$. Let us denote by $\mathcal{W}_n$ the set of players that are computation-saturated and by $\mathcal{B}_n$ the set of players that are communication-saturated on a given arbitrary worker n.

1. If $\sum_k \frac{C_n}{c_k} \leqslant K$ then $\mathcal{W}_n = \emptyset$ and $\forall k, \rho_{n,k}^{(nc)} = \dfrac{B_n}{K.b_k}$.
2. Else, if $\sum_k \frac{c_k}{C_n} \leqslant K$ then $\mathcal{B}_n = \emptyset$ and $\forall k, \rho_{n,k}^{(nc)} = \dfrac{W_n}{K.w_k}$.
3. Else, $\mathcal{B}_n$ and $\mathcal{W}_n$ are non-empty and there exists an integer $m \in [\![1; K-1]\!]$ such that

$$\frac{c_m}{C_n} < \frac{m - \sum_{k=1}^{m} \frac{c_k}{C_n}}{K - m - \sum_{k=m+1}^{K} \frac{C_n}{c_k}} < \frac{c_{m+1}}{C_n}.$$

Then, we have $\mathcal{W}_n = \{1, \ldots, m\}$ and $\mathcal{B}_n = \{m+1, \ldots, K\}$ and

$$\begin{cases} \rho_{n,k}^{(nc)} = \frac{B_n}{b_k} \frac{|\mathcal{W}_n| - \sum_{p \in \mathcal{W}_n} \frac{c_p}{C_n}}{|\mathcal{W}_n||\mathcal{B}_n| - \sum_{p \in \mathcal{W}_n} c_p \sum_{p \in \mathcal{B}_n} \frac{1}{c_p}} & \text{if } k \in \mathcal{B}_n \\ \rho_{n,k}^{(nc)} = \frac{W_n}{w_k} \frac{|\mathcal{B}_n| - \sum_{p \in \mathcal{B}_n} \frac{C_n}{c_p}}{|\mathcal{W}_n||\mathcal{B}_n| - \sum_{p \in \mathcal{W}_n} c_p \sum_{p \in \mathcal{B}_n} \frac{1}{c_p}} & \text{if } k \in \mathcal{W}_n \end{cases}$$

Characterizing the Nash Equilibrium

Theorem 1.

For a given system (N, B, W, K, b, w) there exists exactly one Nash Equilibrium and it can be analytically computed.

Proof.

Under the non-cooperative assumption, on a given worker, an application is either communication-saturated or computation-saturated. Puting schedules in some canonical form enables to determine for each processor, which applications are communication-saturated and which ones are computation-saturated and then to derive the corresponding rates. □

Outline

Pareto Optimality

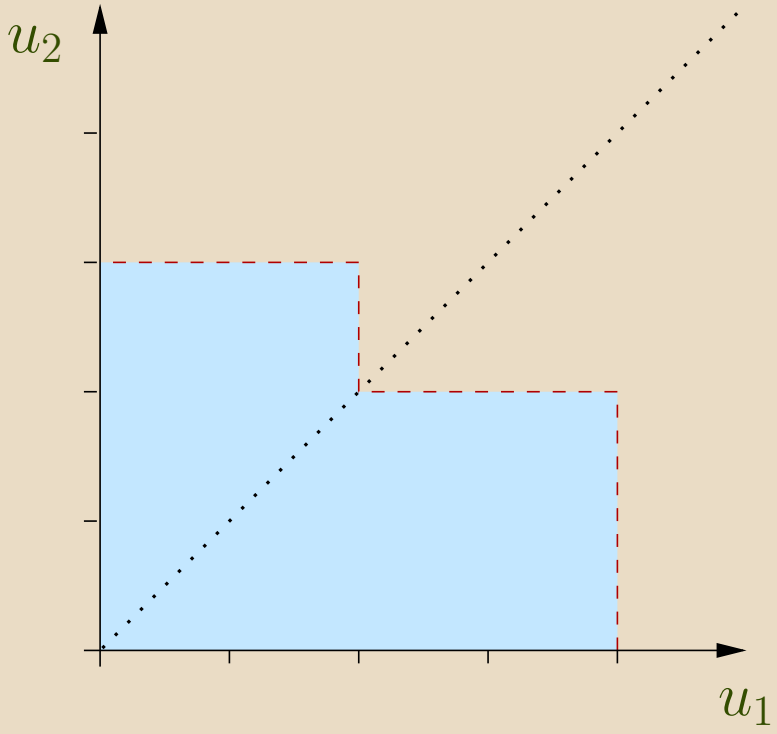
Definition: Pareto optimality.

An allocation is said to be Pareto-optimal if it is impossible to strictly increase the throughput of an application without strictly decreasing the one of another.

Pareto Optimality

Definition: Pareto optimality.

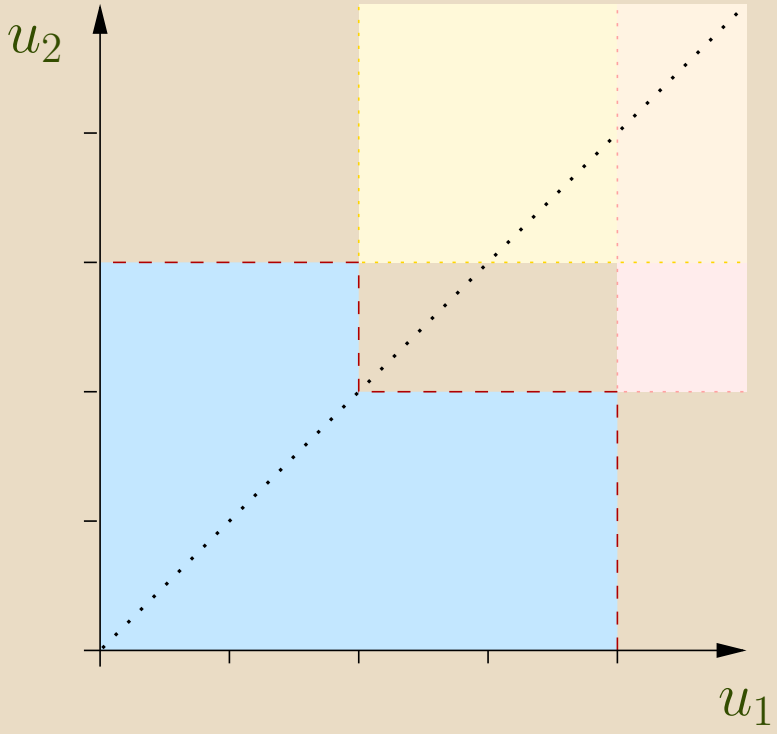
An allocation is said to be Pareto-optimal if it is impossible to strictly increase the throughput of an application without strictly decreasing the one of another.



Pareto Optimality

Definition: Pareto optimality.

An allocation is said to be Pareto-optimal if it is impossible to strictly increase the throughput of an application without strictly decreasing the one of another.



Pareto Optimality

Definition: Pareto optimality.

An allocation is said to be Pareto-optimal if it is impossible to strictly increase the throughput of an application without strictly decreasing the one of another.

When is our Nash Equilibrium Pareto-optimal ?

Theorem 2.

The allocation at the Nash equilibrium is Pareto inefficient if and only if there exists two workers, namely n_1 and n_2 such that all applications are communication-saturated on n_1 and computation-saturated on n_2 (i.e. $\sum_k \frac{B_{n_1}}{W_{n_1}} \frac{w_k}{b_k} \leqslant K$ and $\sum_k \frac{b_k}{w_k} \frac{W_{n_2}}{B_{n_2}} \leqslant K$).

Corrolary: on a single-processor system, the allocation at the Nash equilibrium is Pareto optimal.

Braess-like Paradox

Pareto-inefficient equilibria can exhibit unexpected behavior.

Definition: Braess Paradox [Braess68].

There is a Braess Paradox if there exists two systems ini and aug such that

$$ini < aug \text{ and } \rho^{(nc)}(ini) > \rho^{(nc)}(aug).$$

Braess-like Paradox

Pareto-inefficient equilibria can exhibit unexpected behavior.

Definition: Braess Paradox [Braess68].

There is a Braess Paradox if there exists two systems ini and aug such that

$$ini < aug \text{ and } \rho^{(nc)}(ini) > \rho^{(nc)}(aug).$$

Theorem 3.

In the non-cooperative multi-port scheduling problem, Braess like paradoxes cannot occur.

Proof.

- Defining an equivalence relation on sub-systems.
- Defining an order relation on equivalent sub-systems.

□

Definition [Koutsoupias.Papadimitriou'98] Price of Anarchy:

$$\phi_\Sigma = \max_S \frac{\sum_k \rho_k^{(\Sigma)}(S)}{\sum_k \rho_k^{(nc)}(S)} \geqslant 1.$$

Definition [Koutsoupias.Papadimitriou'98] More generally:

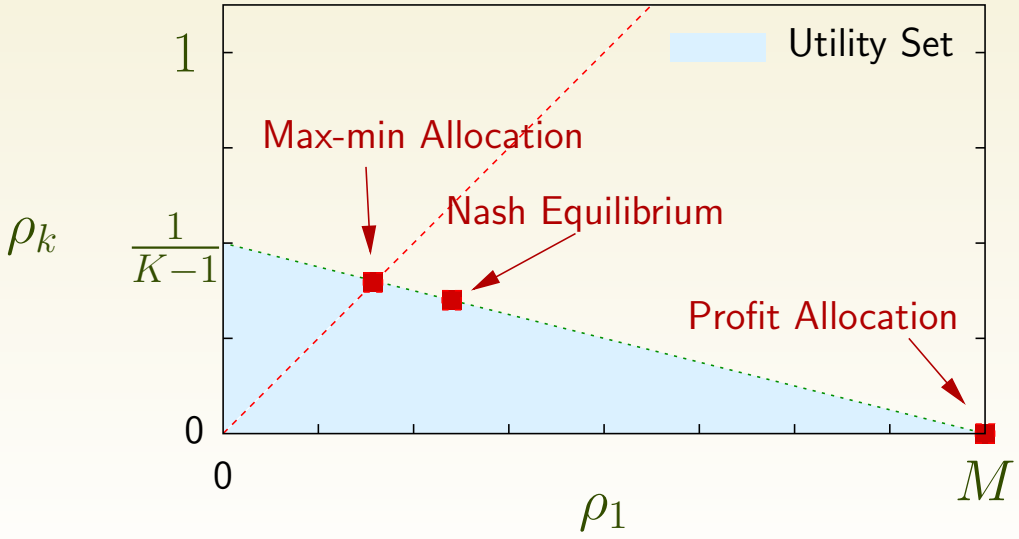
$$I_f(S) = \frac{f\left(\rho_1^{(f)}(S), \ldots, \rho_K^{(f)}(S)\right)}{f\left((\rho_1^{(nc)}(S), \ldots, \rho_K^{(nc)}(S)\right)} \geqslant 1.$$

Definition [Koutsoupias.Papadimitriou'98] More generally:

$$I_f(S) = \frac{f\left(\rho_1^{(f)}(S), \ldots, \rho_K^{(f)}(S)\right)}{f\left((\rho_1^{(nc)}(S), \ldots, \rho_K^{(nc)}(S)\right)} \geqslant 1.$$

Problem measures the "distance" to a particular point...

Illustration 1 machine ($B_1 = W_1 = 1$) and K applications $b = (\frac{1}{M}, 1...1)$ and $w = (\frac{1}{M}, 1...1)$.



$$I_\Sigma(S_{M,K}) = \frac{M}{\frac{M}{K} + \frac{K-1}{K}} \xrightarrow[M\to\infty]{} K$$

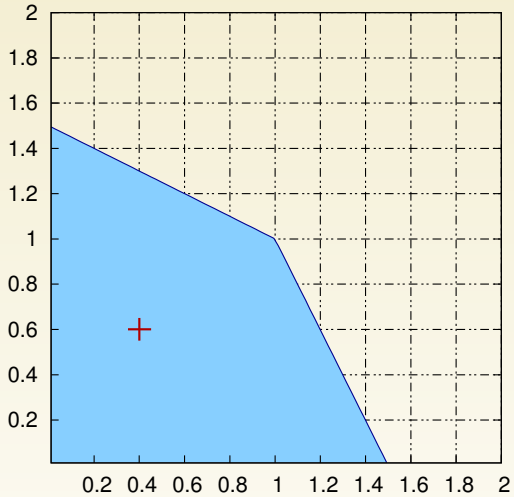
Utility set and allocations $S_{M,K}$ ($K = 3, M = 2$).

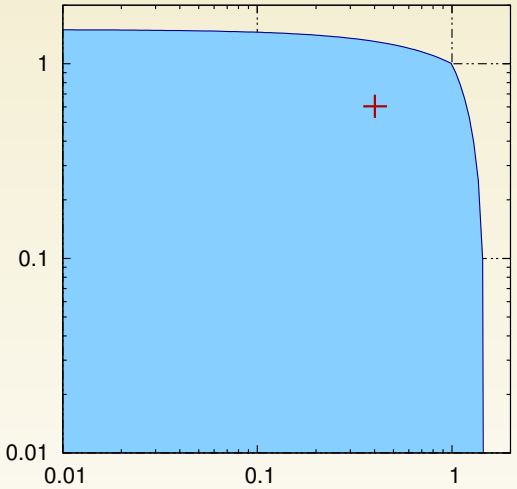
Rationale measure the "distance" to the Pareto set.

$$\phi = \max_S I(S) = \exp(d_\infty(\log(\rho_{n,*}^{(nc)}(S), \log(\overline{\mathcal{P}}(S)))$$

$$= \max_S \min_{\rho \in \overline{\mathcal{P}}(S)} \max_k \max \left(\frac{\rho_{n,k}^{(nc)}(S)}{\rho_k}, \frac{\rho_k}{\rho_{n,k}^{(nc)}(S)} \right)$$

Illustration





Degradation Factor is related to ε-approximation of Pareto-curves [Papadimitriou.Yannakakis'00].

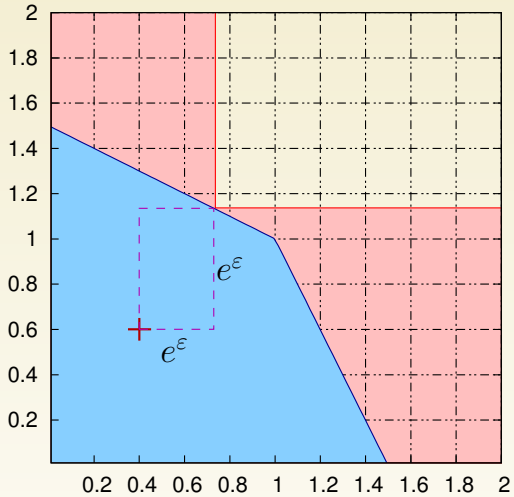
In our context, Selfishness Degradation Factor is at least 2.

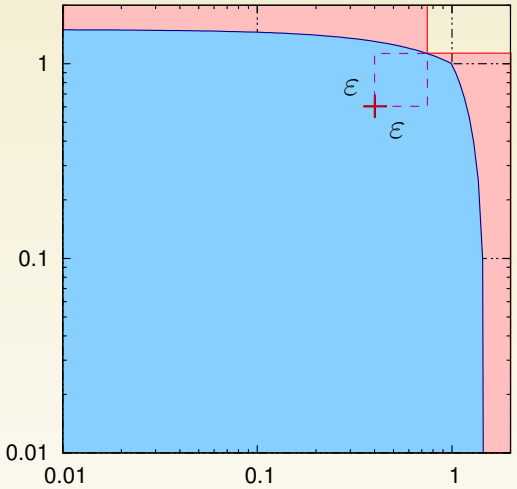
Rationale measure the "distance" to the Pareto set.

$$\phi = \max_S I(S) = \exp(d_\infty(\log(\rho_{n,*}^{(nc)}(S), \log(\overline{\mathcal{P}}(S)))$$

$$= \max_S \min_{\rho \in \overline{\mathcal{P}}(S)} \max_k \max \left(\frac{\rho_{n,k}^{(nc)}(S)}{\rho_k}, \frac{\rho_k}{\rho_{n,k}^{(nc)}(S)} \right)$$

Illustration

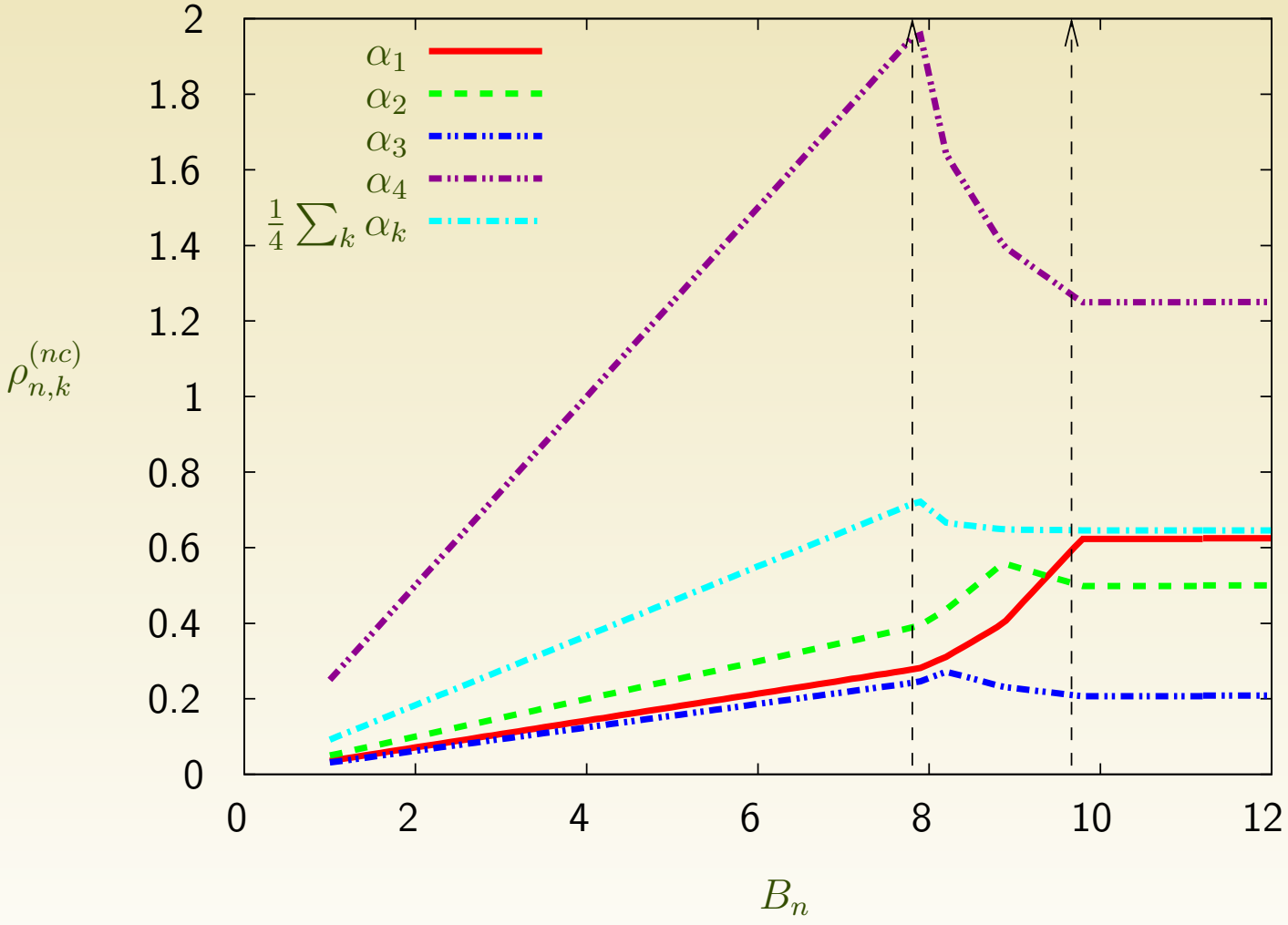




Degradation Factor is related to ε-approximation of Pareto-curves [Papadimitriou.Yannakakis'00].

In our context, Selfishness Degradation Factor is at least 2.

Pareto Optimality and Monotonicity of Performance Measures



Most classical performance measures decrease with resource augmentation!

Recap

Conclusion

- Applying fair and optimal sharing on each resource does not ensure any fairness nor efficiency when users do not cooperate.
- Being "locally Pareto optimal" (i.e on each single machine) does not help being Pareto optimal.
- Even with Pareto optimal situations, classical performance measures can be non monotonic.

$\rightsquigarrow$ either applications cooperate or new complex and global access policies should be designed

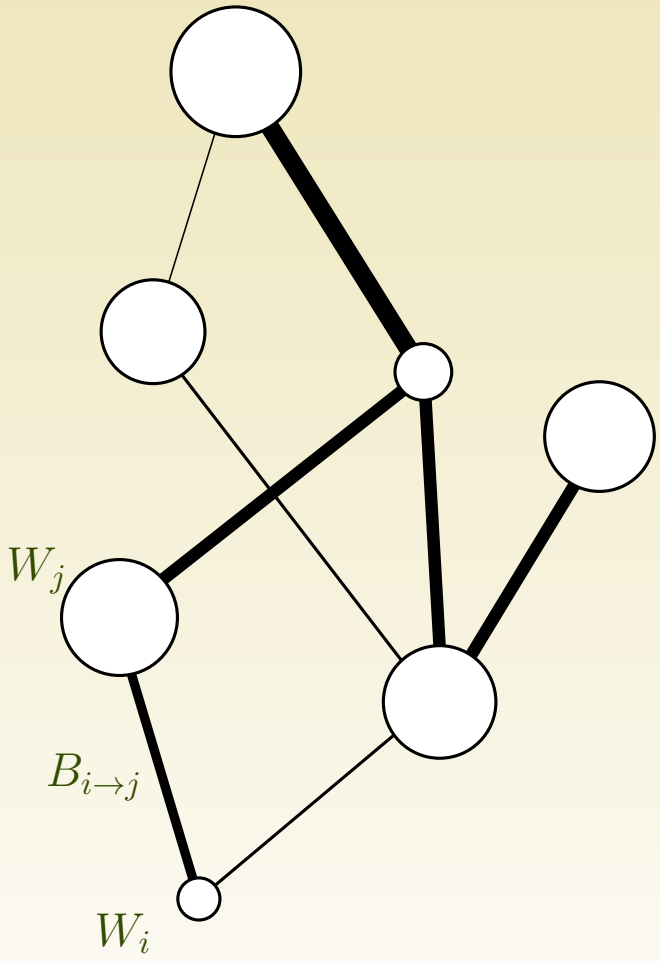
Other Future Work

- Measuring Pareto-inefficiency is an open question.
- In the one-port communication model, Braess-like paradoxes seem to arise.

Outline

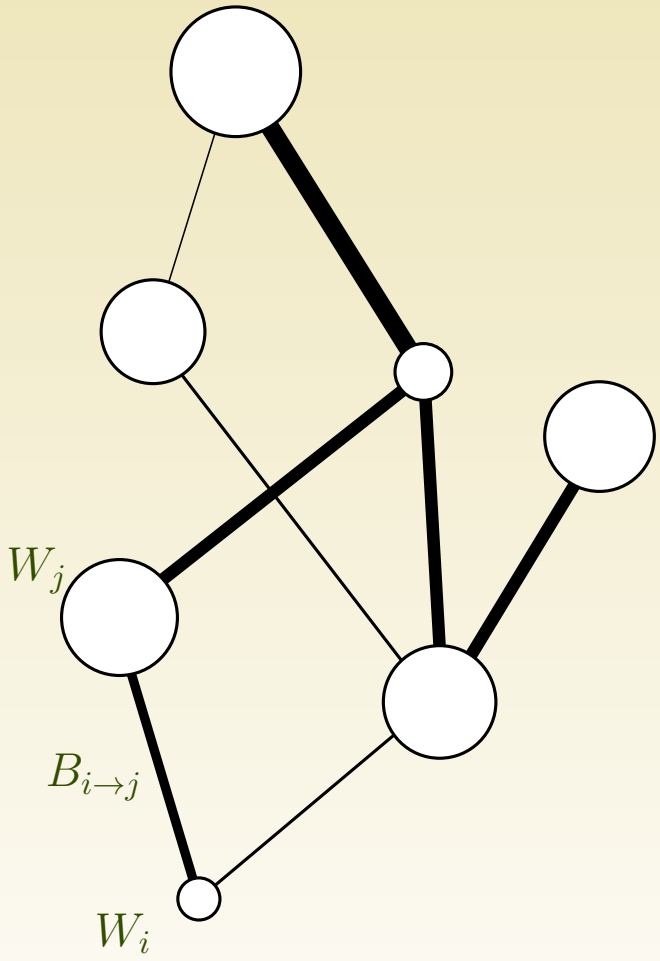
Outline

Platform Model



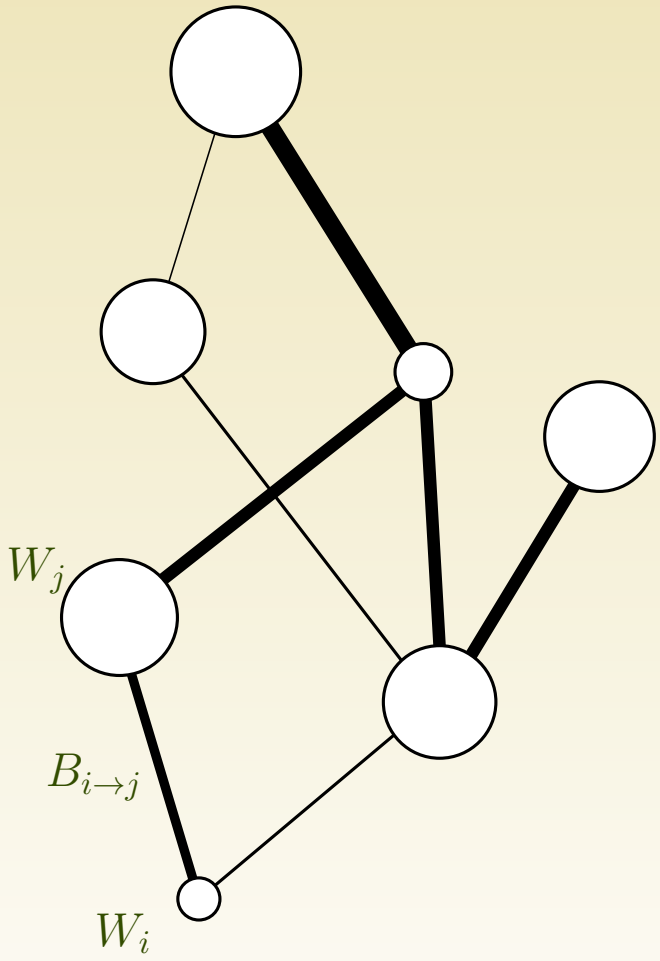
- General platform graph $G = (N, E, W, B)$

Platform Model



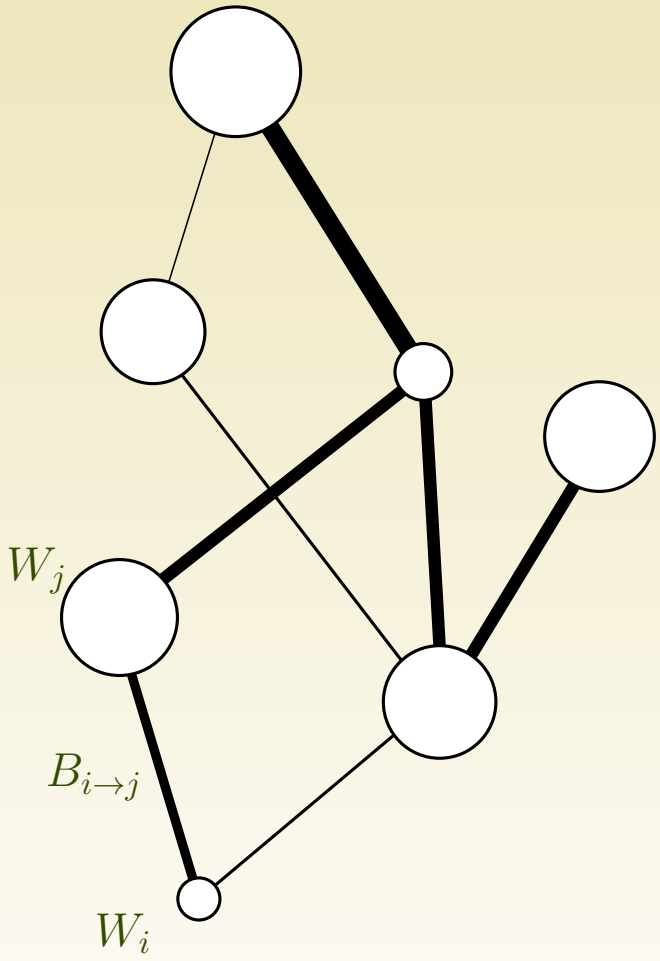
- General platform graph $G = (N, E, W, B)$
- Speed of $P_n \in N$: W_n (in MFlops/s)

Platform Model



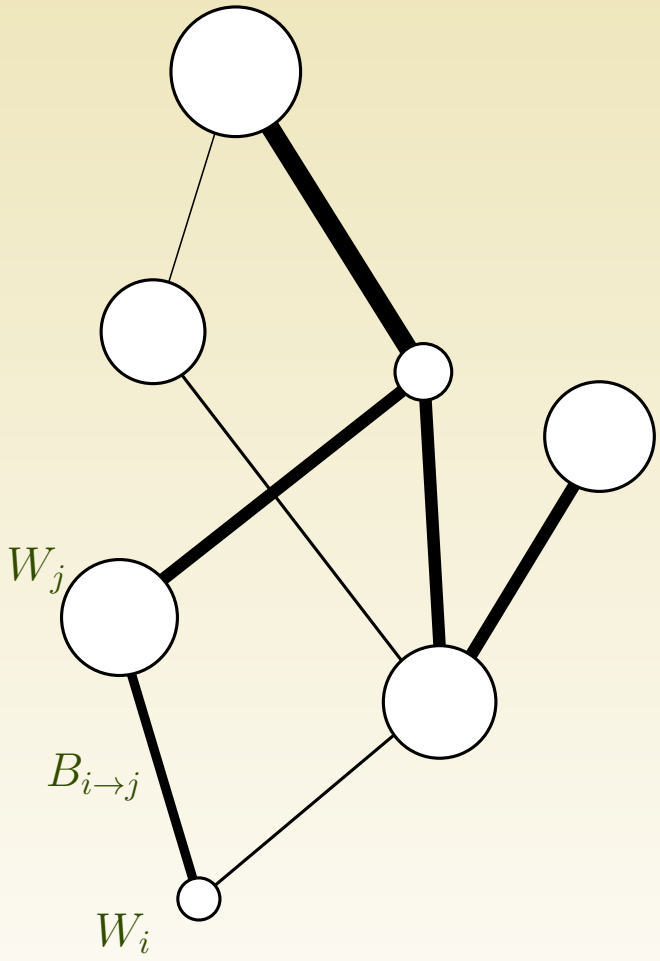
- General platform graph $G = (N, E, W, B)$
- Speed of $P_n \in N$: W_n (in MFlops/s)
- Bandwidth of $(P_i \to P_j)$: $B_{i,j}$ (in MB/s)

Platform Model



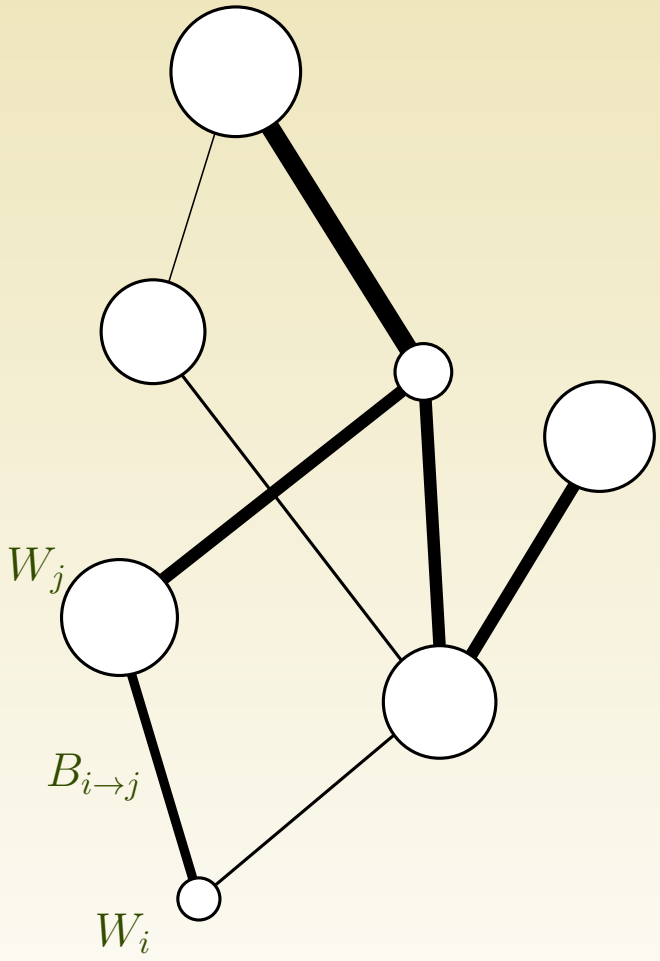
- General platform graph $G = (N, E, W, B)$
- Speed of $P_n \in N$: W_n (in MFlops/s)
- Bandwidth of $(P_i \to P_j)$: $B_{i,j}$ (in MB/s)
- Linear-cost communication and computation model: $X/B_{i,j}$ time units to send a message of size X from P_i to P_j.

Platform Model



- General platform graph $G = (N, E, W, B)$
- Speed of $P_n \in N$: W_n (in MFlops/s)
- Bandwidth of $(P_i \to P_j)$: $B_{i,j}$ (in MB/s)
- Linear-cost communication and computation model: $X/B_{i,j}$ time units to send a message of size X from P_i to P_j.
- Communications and computations can be overlapped.

Platform Model

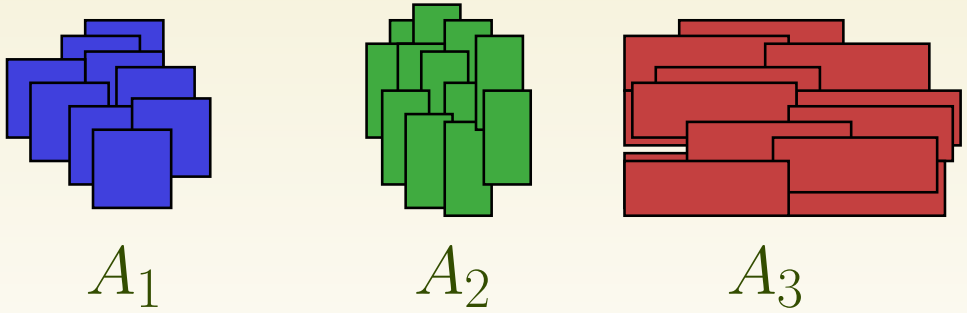


- General platform graph $G = (N, E, W, B)$
- Speed of $P_n \in N$: W_n (in MFlops/s)
- Bandwidth of $(P_i \to P_j)$: $B_{i,j}$ (in MB/s)
- Linear-cost communication and computation model: $X/B_{i,j}$ time units to send a message of size X from P_i to P_j.
- Communications and computations can be overlapped.
- Multi-port communication model.

Application Model

Multiple applications:

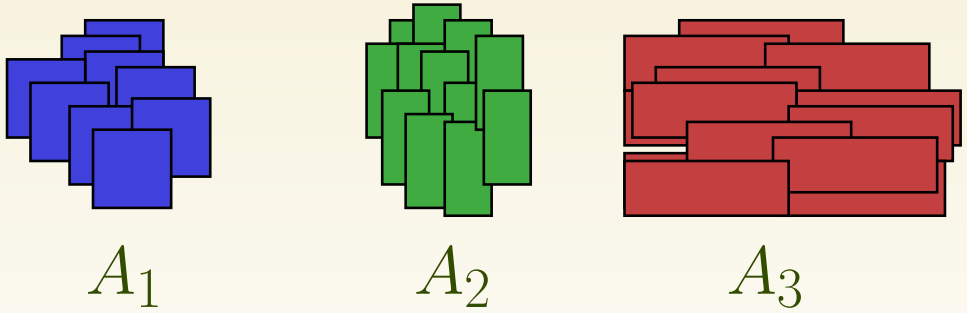
- A set A of K applications $A_1, \ldots, A_K$



Application Model

Multiple applications:

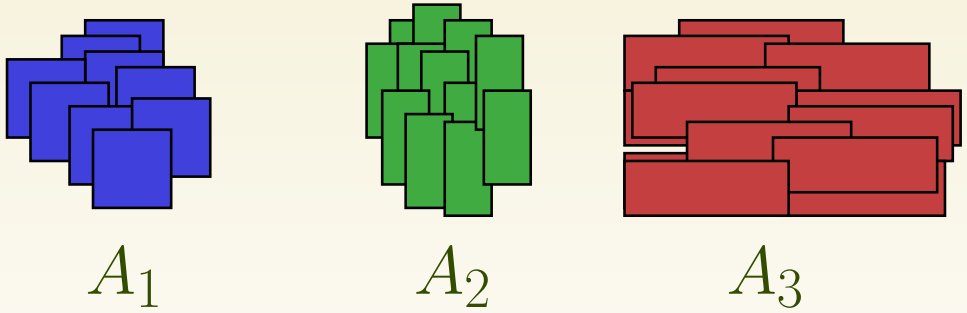
- A set A of K applications $A_1, \ldots, A_K$
- Each consisting in a large number of same-size independent tasks $\rightsquigarrow$ each application is defined by a communication cost w_k (in MFlops) and a communication cost b_k (in MB)



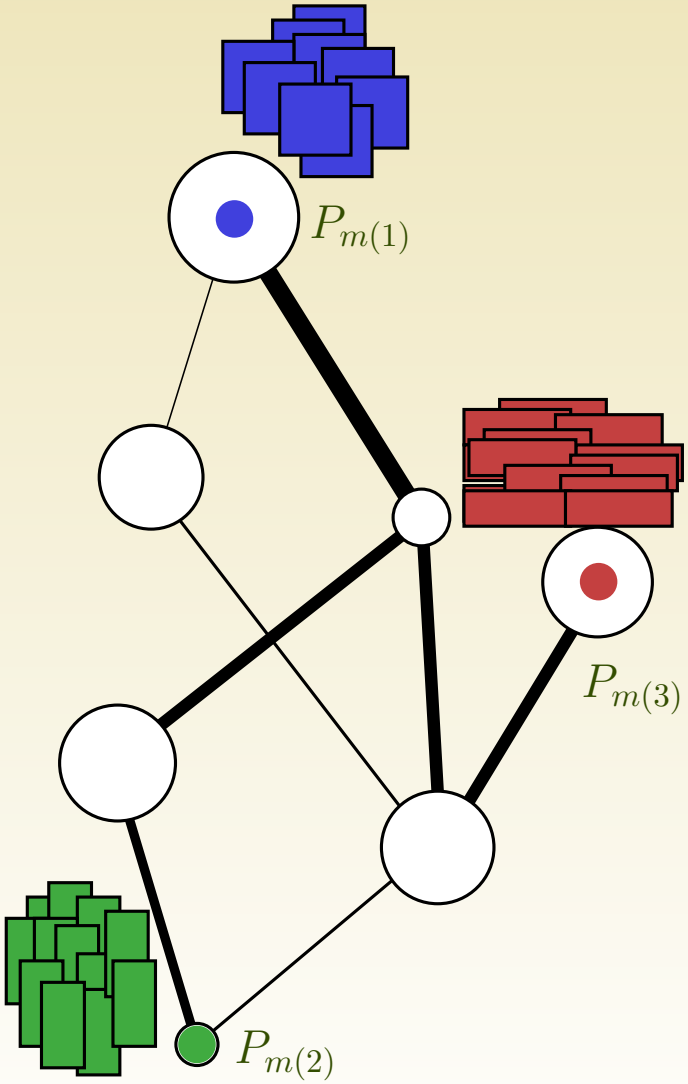
Application Model

Multiple applications:

- A set A of K applications $A_1, \ldots, A_K$
- Each consisting in a large number of same-size independent tasks $\rightsquigarrow$ each application is defined by a communication cost w_k (in MFlops) and a communication cost b_k (in MB)
- Different communication and computation demands for different applications

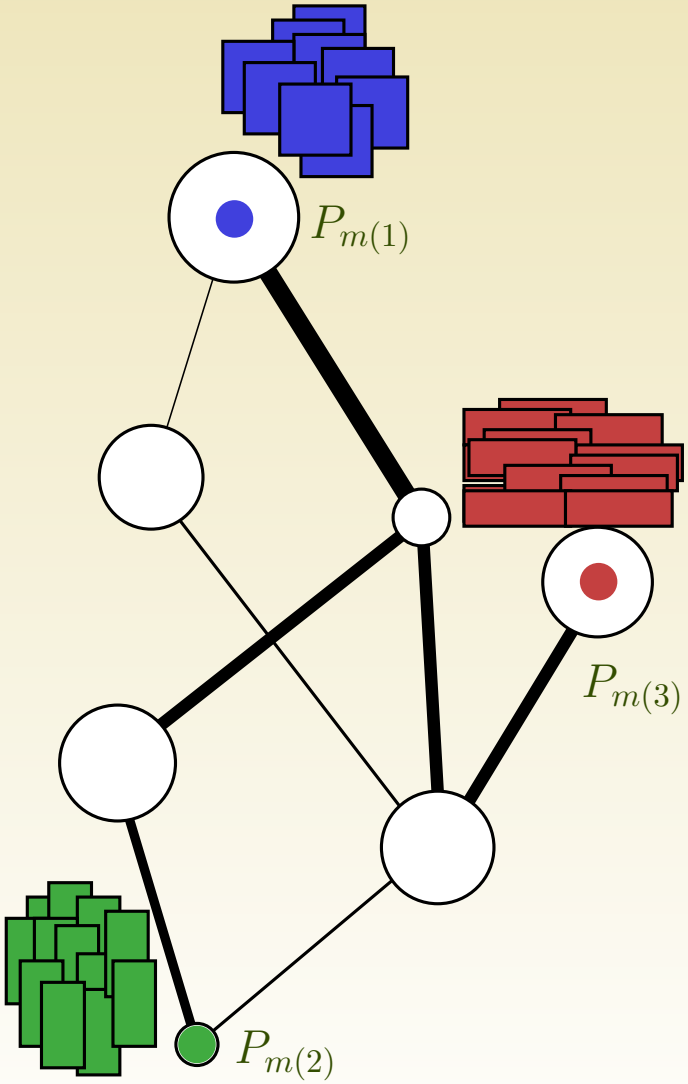


Hierarchical Deployment



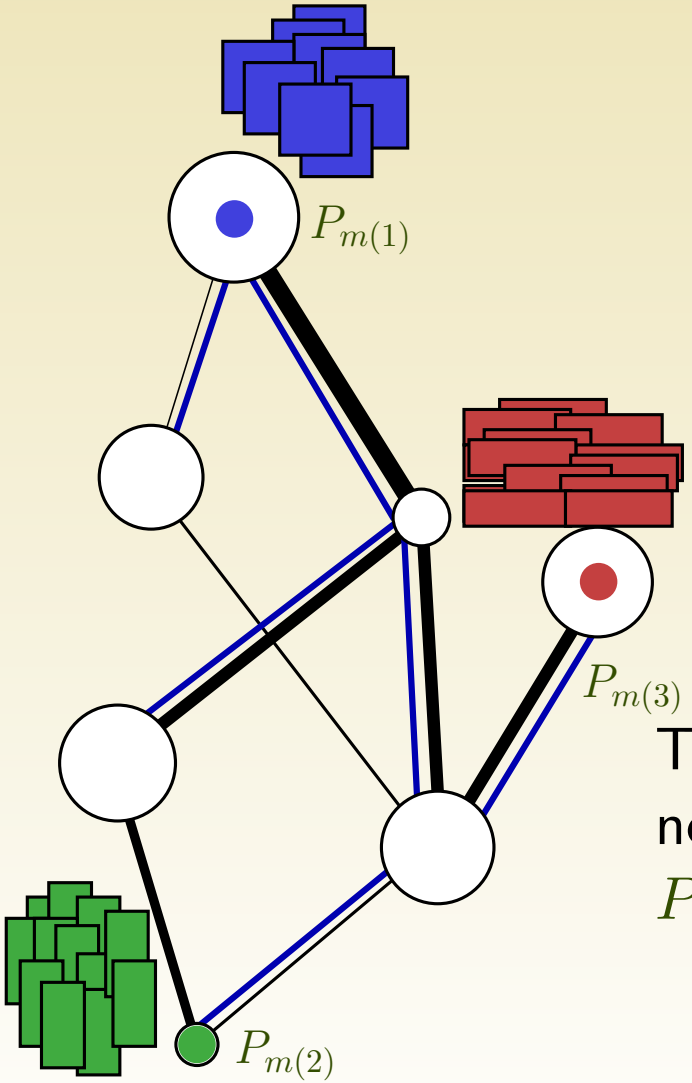
- Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k

Hierarchical Deployment



- Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k
- Communication are only required outwards from the master nodes: the amount of data returned by the worker is negligible

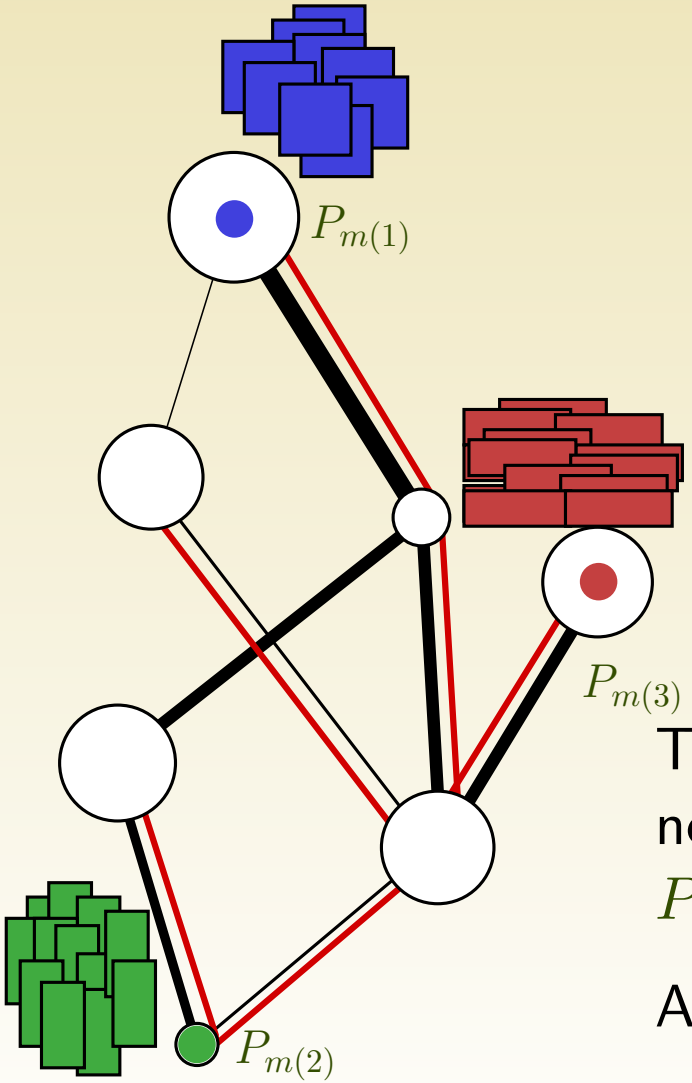
Hierarchical Deployment



- Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k
- Communication are only required outwards from the master nodes: the amount of data returned by the worker is negligible
- Each application A_k is deployed on the platform as a tree

Therefore if an application k wants to use a node P_n, all its data will use a single path from $P_{m(k)}$ to P_n denoted by $(P_{m(k)} \rightsquigarrow P_n)$

Hierarchical Deployment



- Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k
- Communication are only required outwards from the master nodes: the amount of data returned by the worker is negligible
- Each application A_k is deployed on the platform as a tree

Therefore if an application k wants to use a node P_n, all its data will use a single path from $P_{m(k)}$ to P_n denoted by $(P_{m(k)} \rightsquigarrow P_n)$

All deployment trees may be different

Steady-State Scheduling and Utility

- All tasks of a given application are identical and independent
  $\rightsquigarrow$ we do not really need to care about where and when

Steady-State Scheduling and Utility

- All tasks of a given application are identical and independent
  $\rightsquigarrow$ we do not really need to care about where and when
- We only need to focus on average values in steady-state

Steady-State Scheduling and Utility

- All tasks of a given application are identical and independent
  $\rightsquigarrow$ we do not really need to care about where and when
- We only need to focus on average values in steady-state
- Steady-state values:

Steady-State Scheduling and Utility

- All tasks of a given application are identical and independent
  $\rightsquigarrow$ we do not really need to care about where and when
- We only need to focus on average values in steady-state
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor n per time unit: $\rho_{n,k}$

Steady-State Scheduling and Utility

- All tasks of a given application are identical and independent
  $\rightsquigarrow$ we do not really need to care about where and when
- We only need to focus on average values in steady-state
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor n per time unit: $\rho_{n,k}$
  - Throughput of application k : $\rho_k = \sum_{n \in N} \rho_{n,k}$

Steady-State Scheduling and Utility

- All tasks of a given application are identical and independent $\rightsquigarrow$ we do not really need to care about where and when
- We only need to focus on average values in steady-state
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor n per time unit: $\rho_{n,k}$
  - Throughput of application k : $\rho_k = \sum_{n \in N} \rho_{n,k}$
  - Variables need to respect the following constraints

$$\forall n, \quad \sum_k \rho_{n,k} w_k \leqslant W_n$$

$$\forall (P_i \to P_j), \quad \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \rho_{n,k} b_k \leqslant B_{i,j}$$

Steady-State Scheduling and Utility

- All tasks of a given application are identical and independent $\rightsquigarrow$ we do not really need to care about where and when
- We only need to focus on average values in steady-state
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor n per time unit: $\rho_{n,k}$
  - Throughput of application k : $\rho_k = \sum_{n \in N} \rho_{n,k}$
  - Variables need to respect the following constraints

$$\forall n, \quad \sum_k \rho_{n,k} w_k \leqslant W_n$$

$$\forall (P_i \to P_j), \quad \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \rho_{n,k} b_k \leqslant B_{i,j}$$

We would like to maximize all throughputs ρ_k.

"Defining" fairness is one way to go from a multi-criteria problem to a more classical mono-criteria problem.

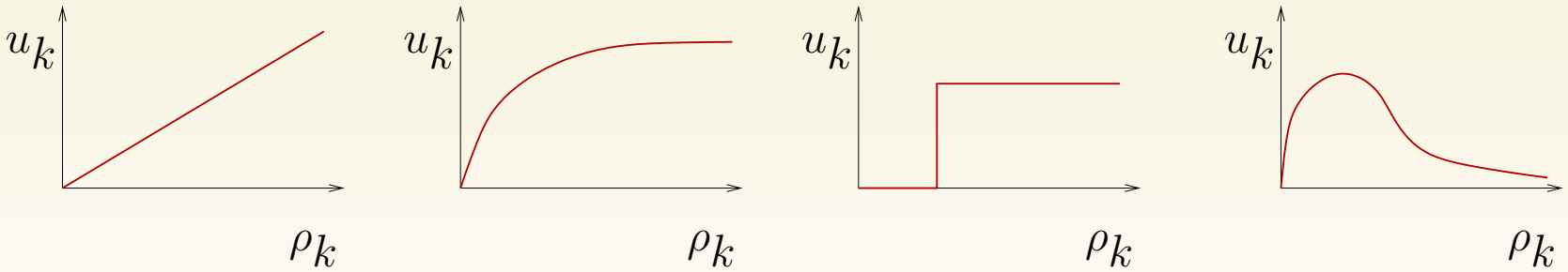
Utility Function

In a general context, each application is characterized by a utility function u_k defined on $(\rho_{n,k})_{1\leqslant k\leqslant K, 1\leqslant n\leqslant N}$.

In our context, the utility is simply defined by

$$u_k(\rho) = \sum_n \rho_{n,k} = \rho_k$$

But we could perfectly imagine other utility functions:

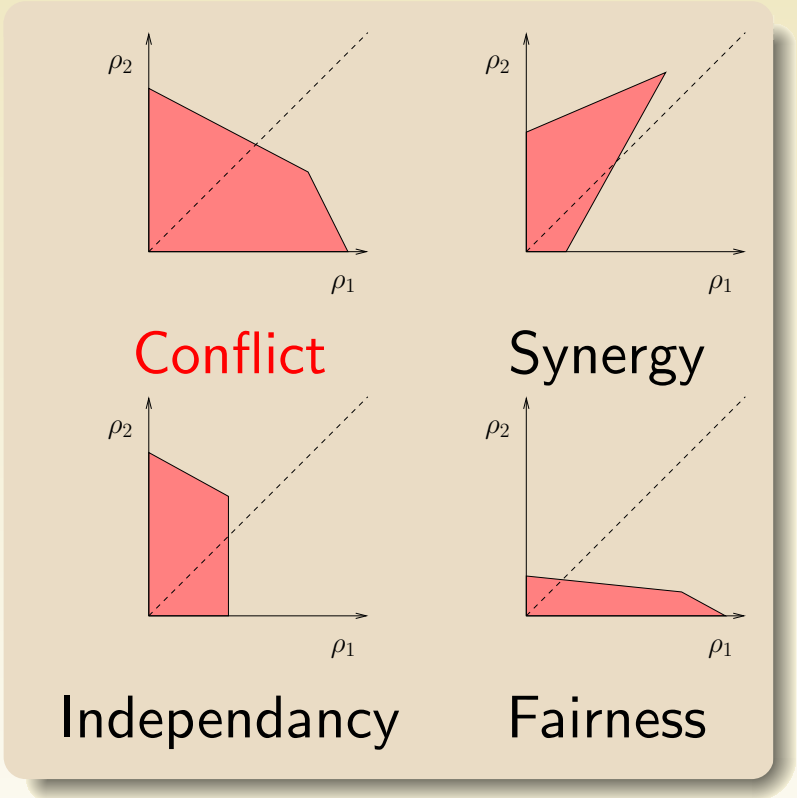


linear | Voice over IP | threshold | price-accounting

Utility Set and Fairness

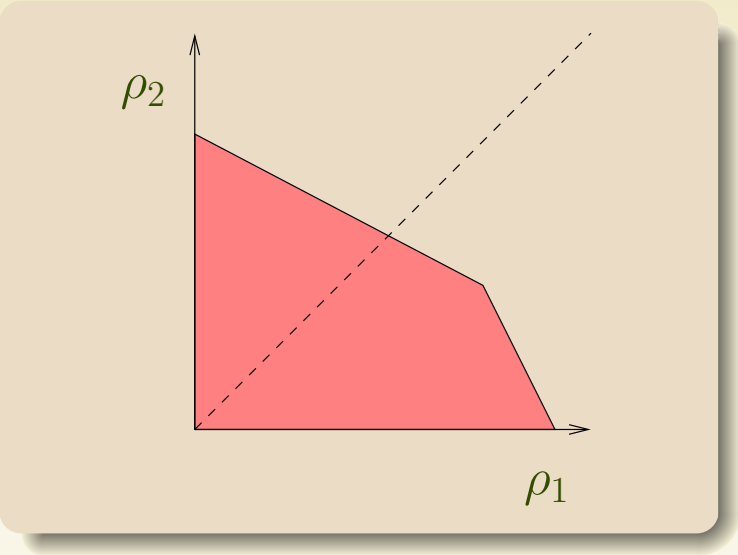
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

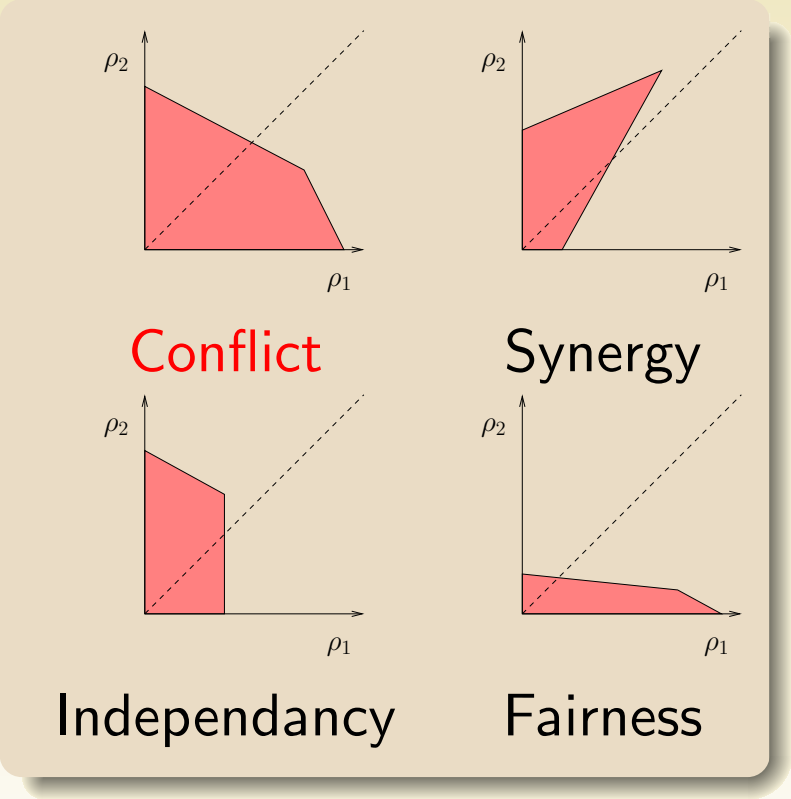
$$\begin{aligned} \rho_{\cdot}^{(1)} c^{(1)} + \rho_{\cdot}^{(2)} c^{(2)} &\leqslant W_u \\ \rho_{\cdot}^{(1)} b^{(1)} + \rho_{\cdot}^{(2)} b^{(2)} &\leqslant B_u \\ \rho_{\cdot}^{(1)} &\geqslant 0 \\ \rho_{\cdot}^{(2)} &\geqslant 0 \end{aligned}$$



Utility Set and Fairness

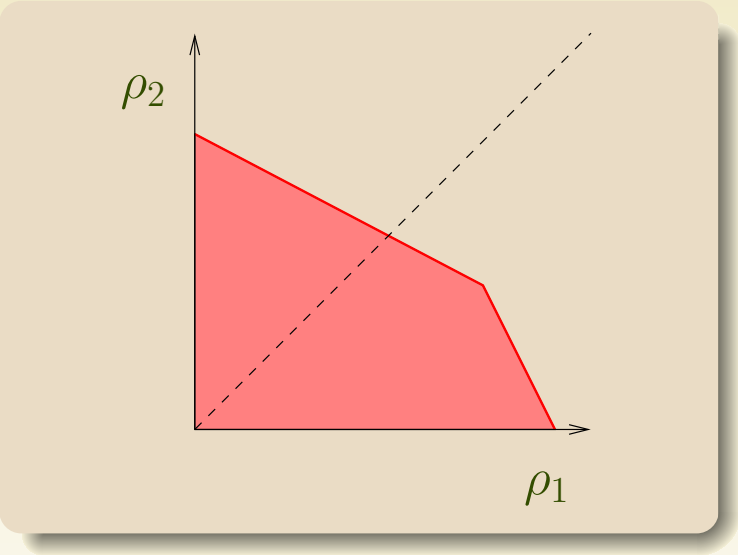
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

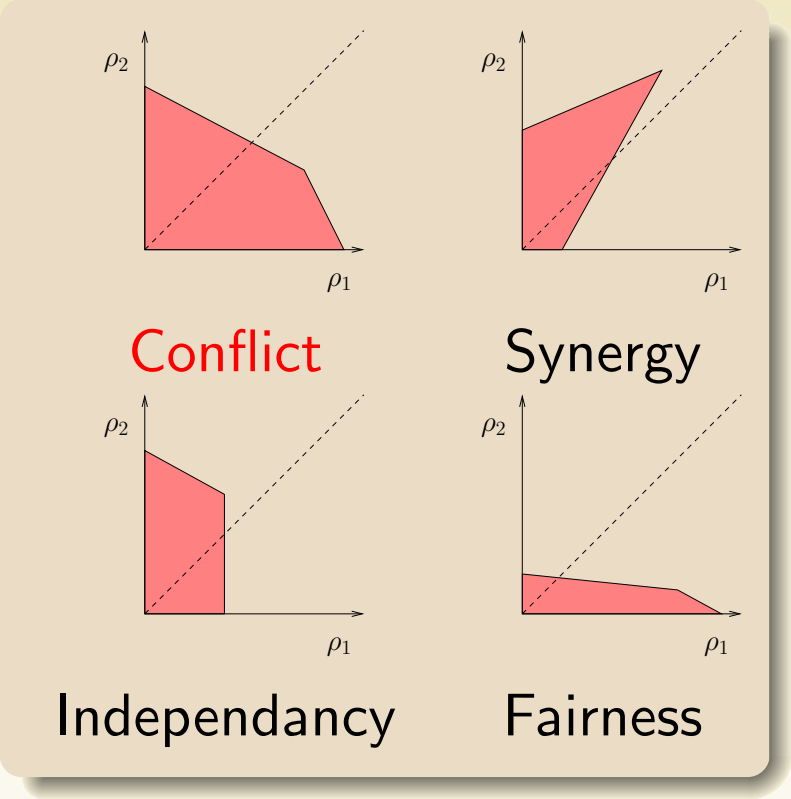




Utility Set and Fairness

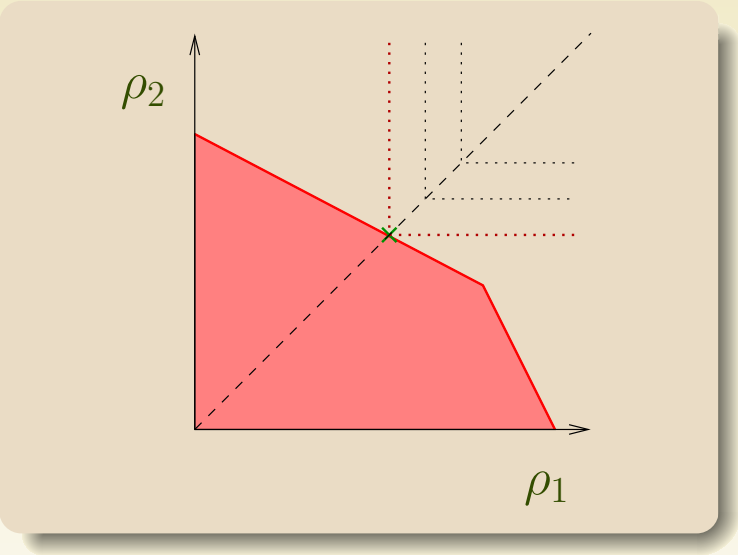
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

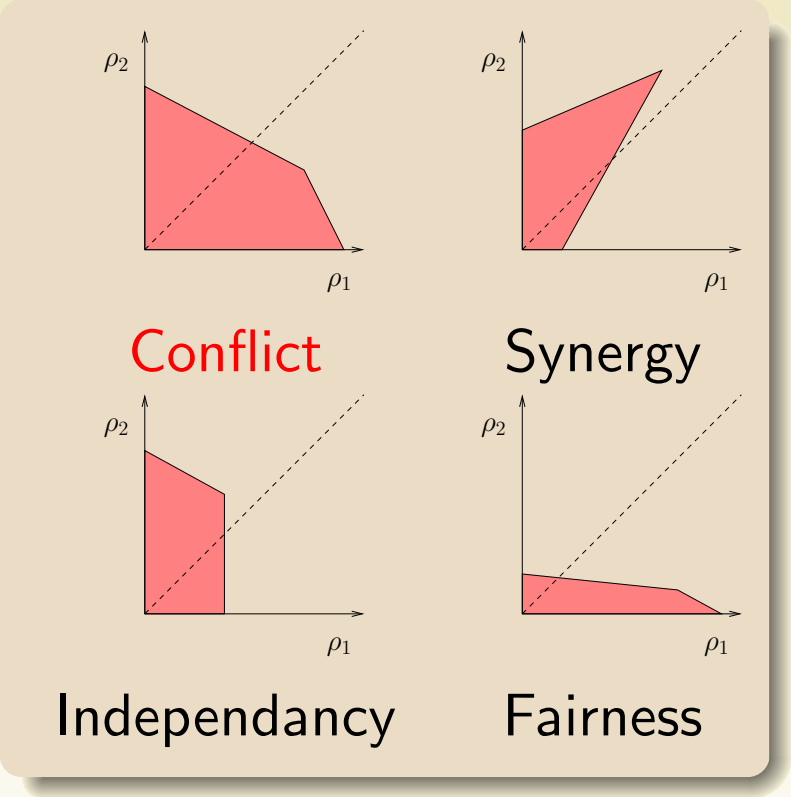




Utility Set and Fairness

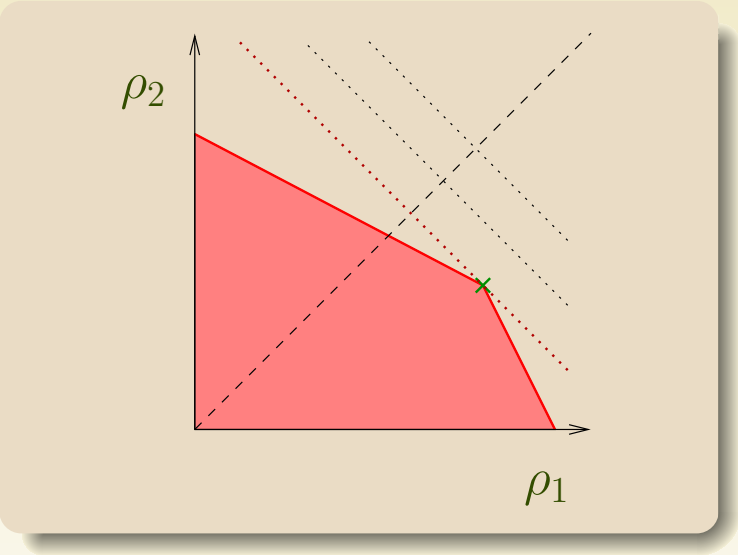
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

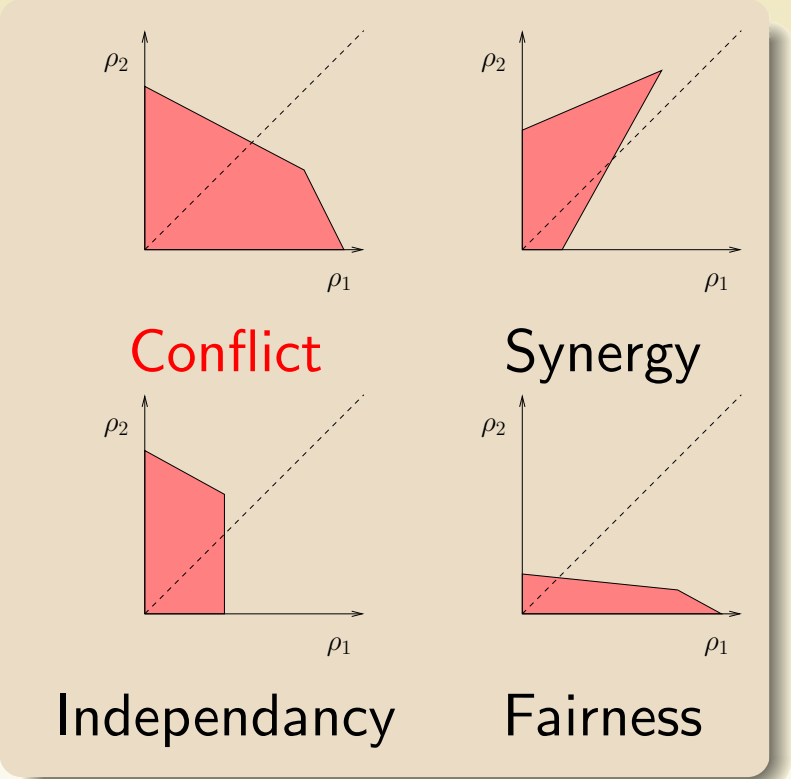




Utility Set and Fairness

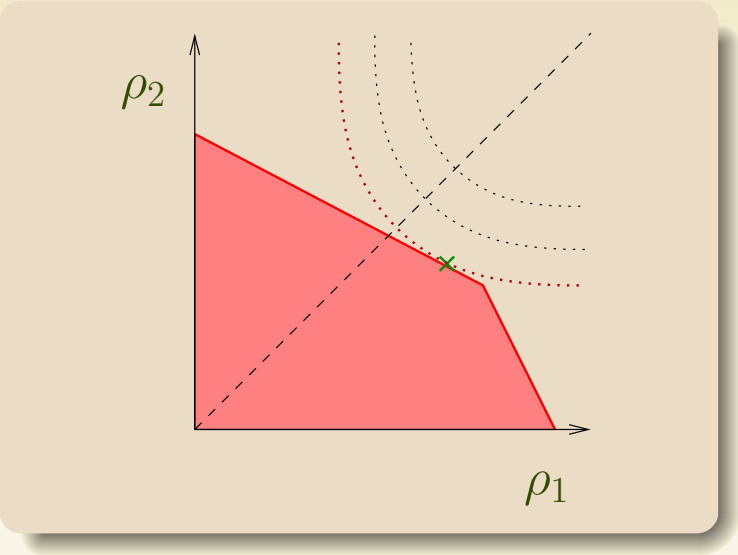
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

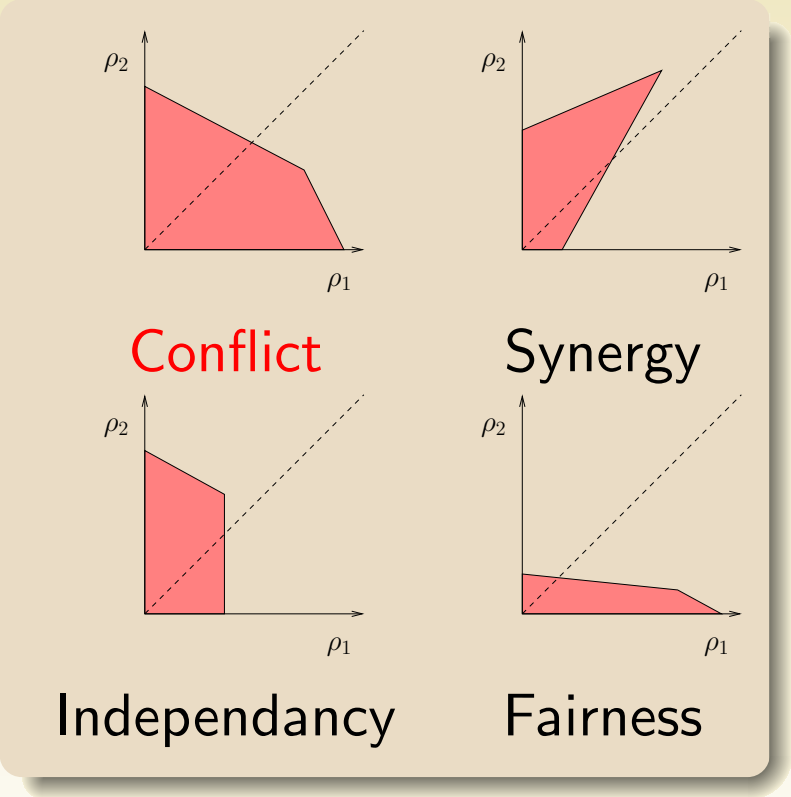




Utility Set and Fairness

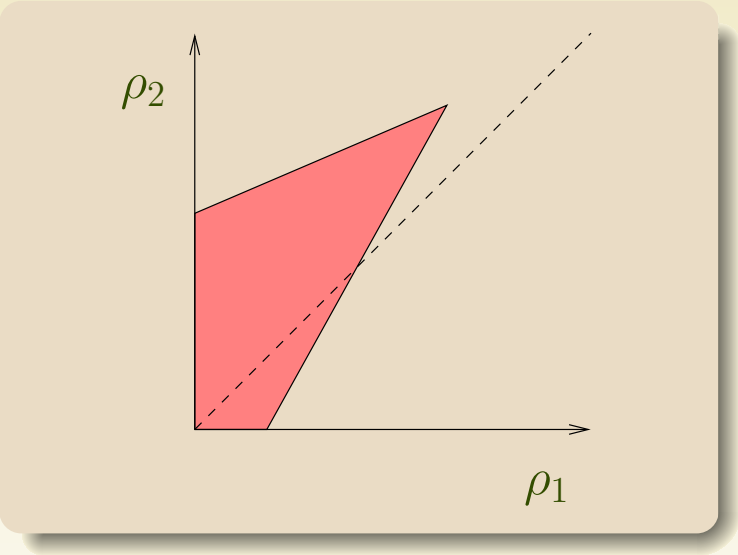
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

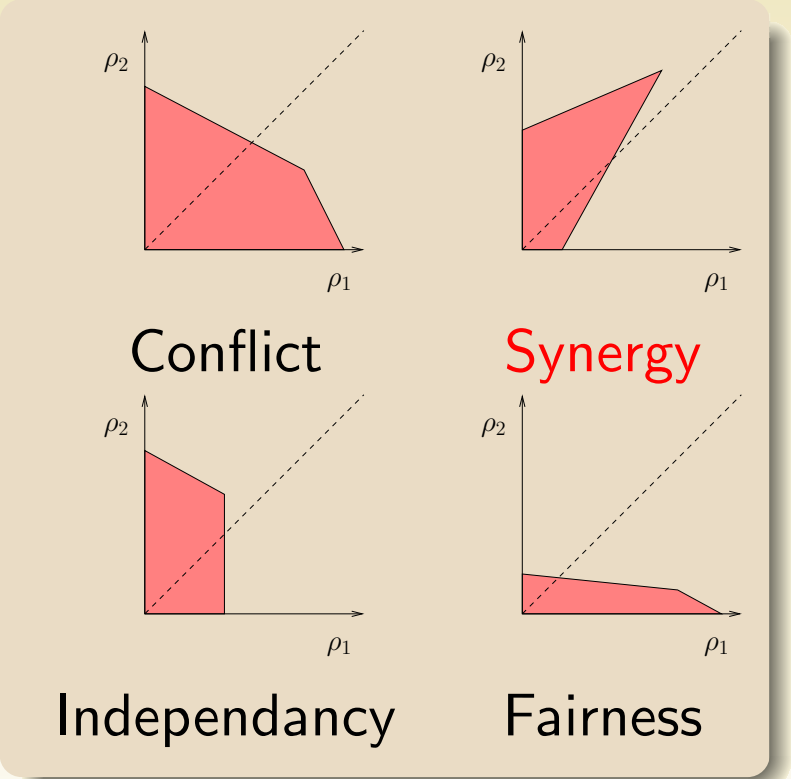




Utility Set and Fairness

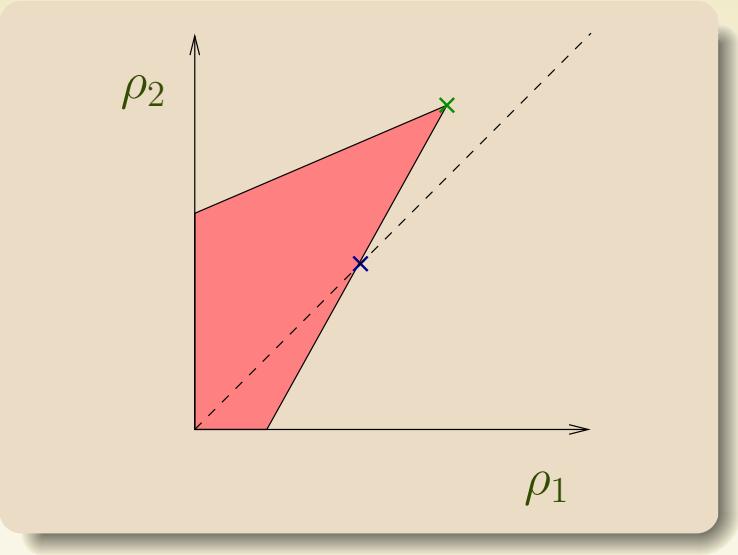
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

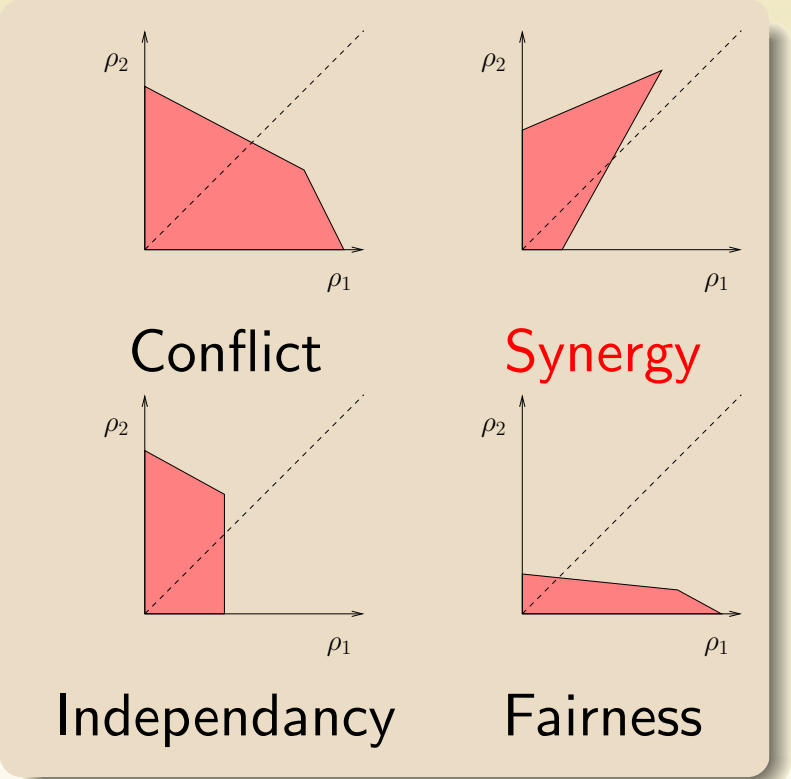




Utility Set and Fairness

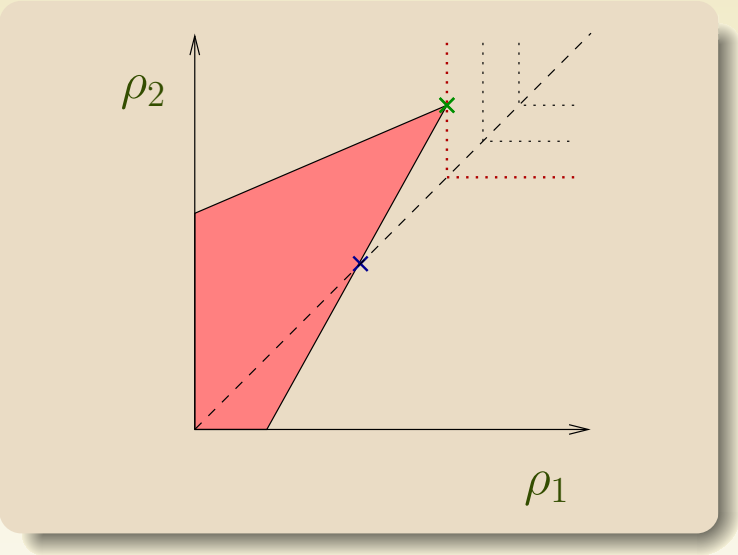
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

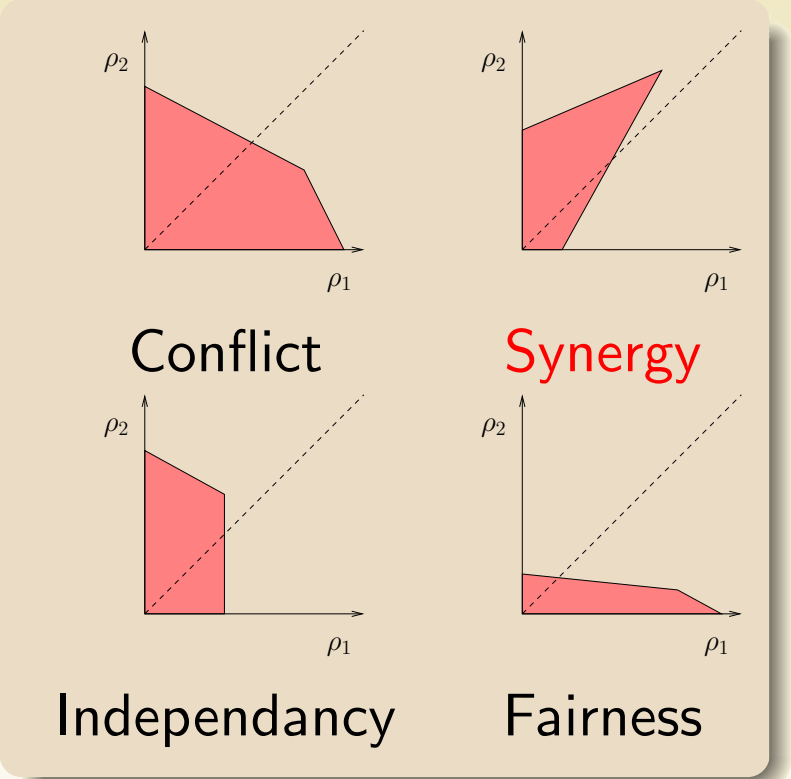




Utility Set and Fairness

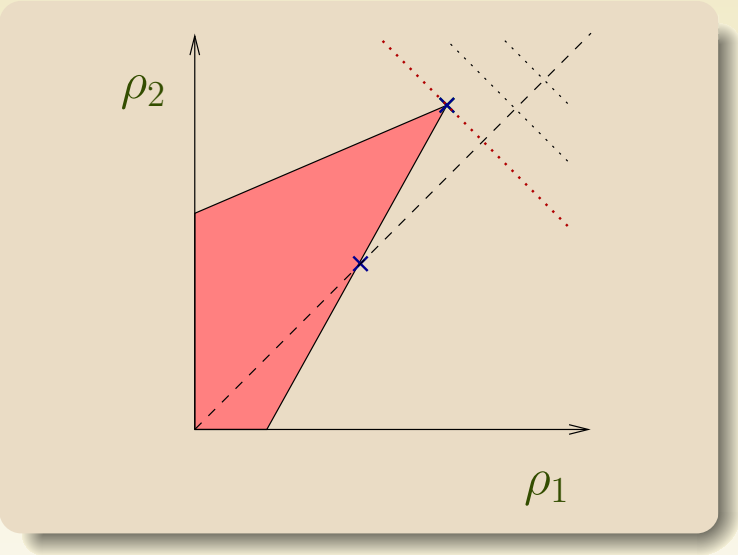
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

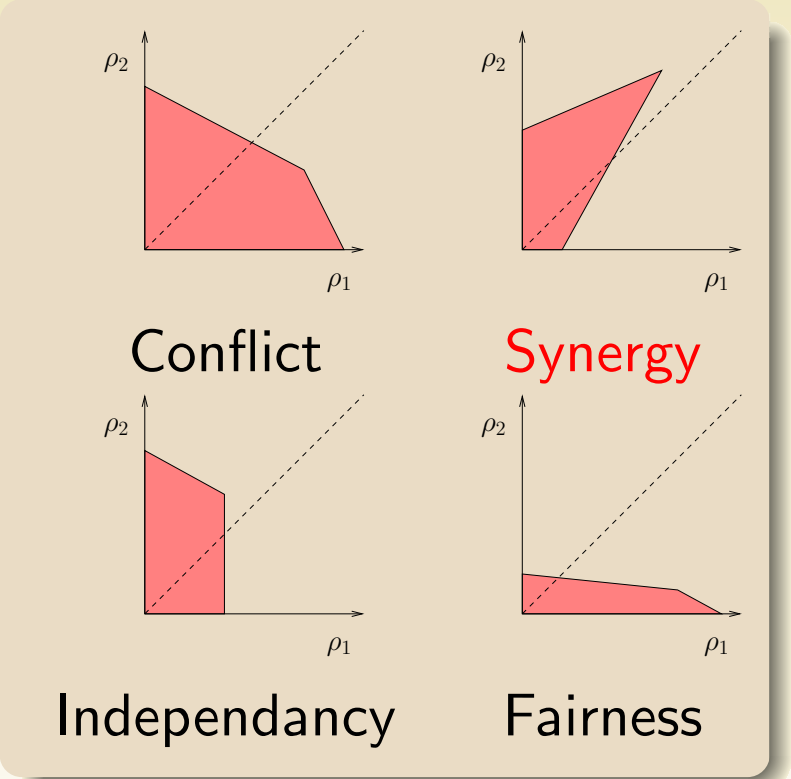




Utility Set and Fairness

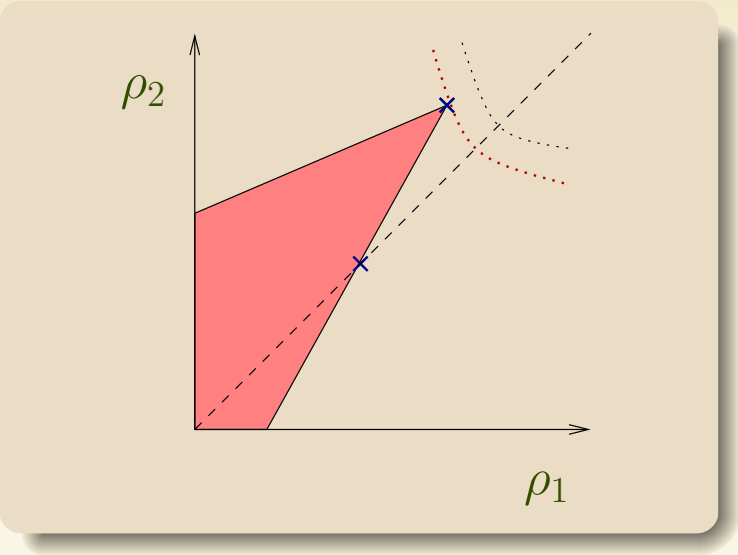
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

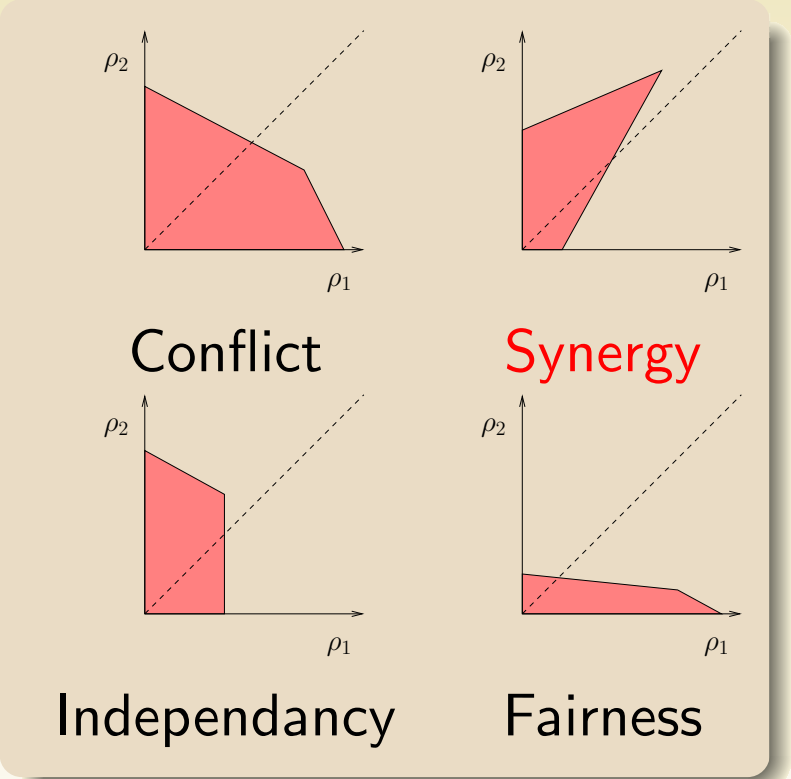




Utility Set and Fairness

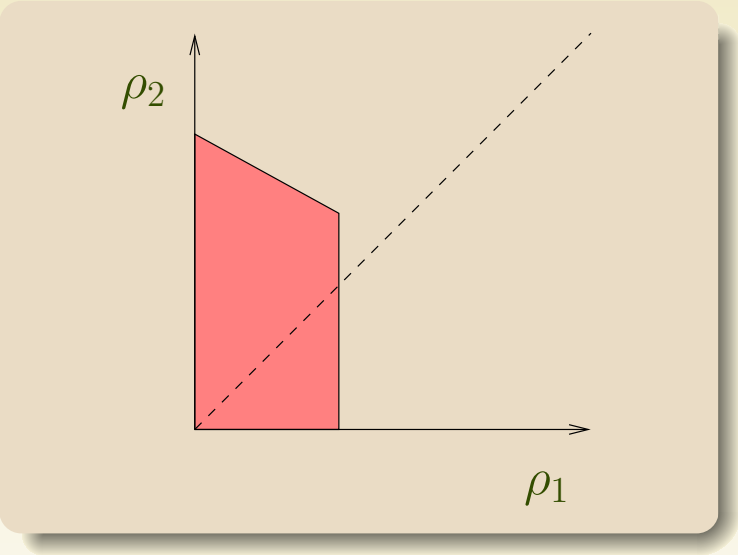
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

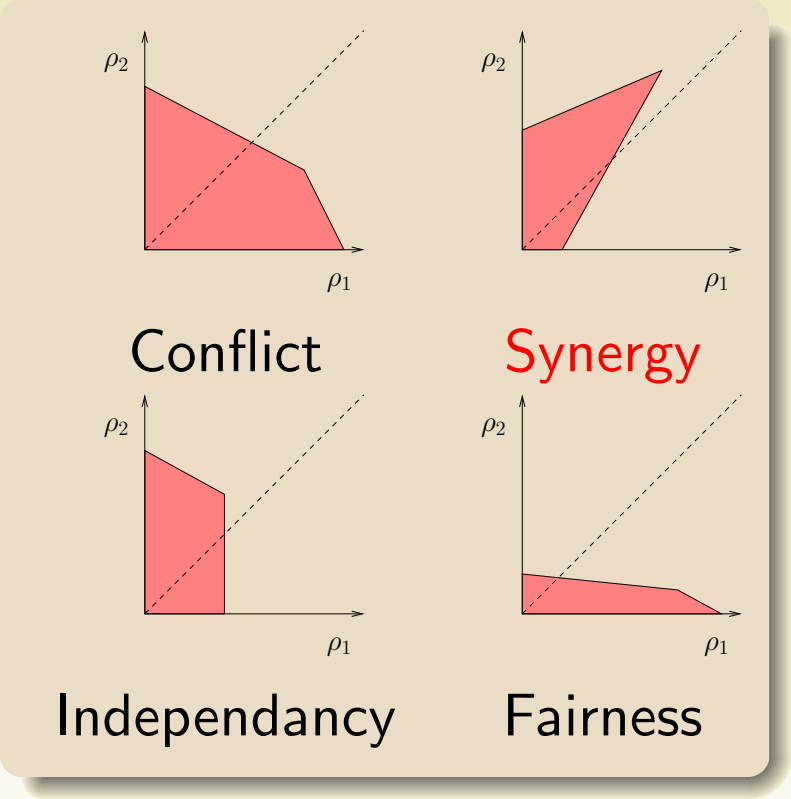




Utility Set and Fairness

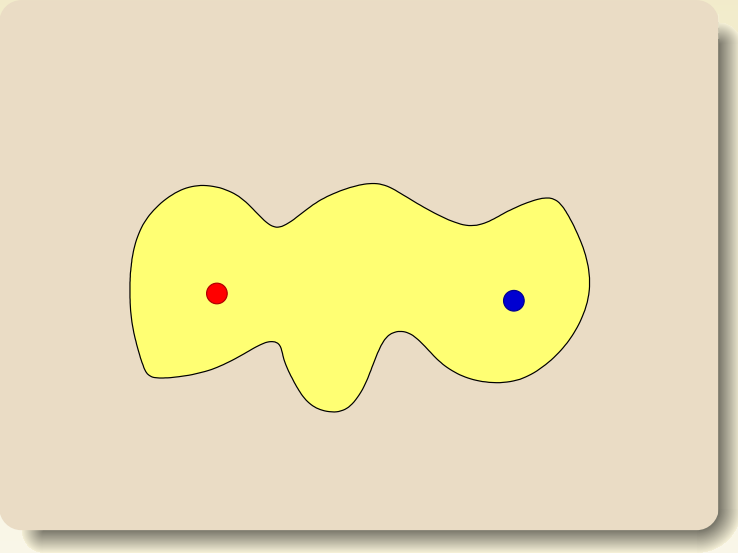
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

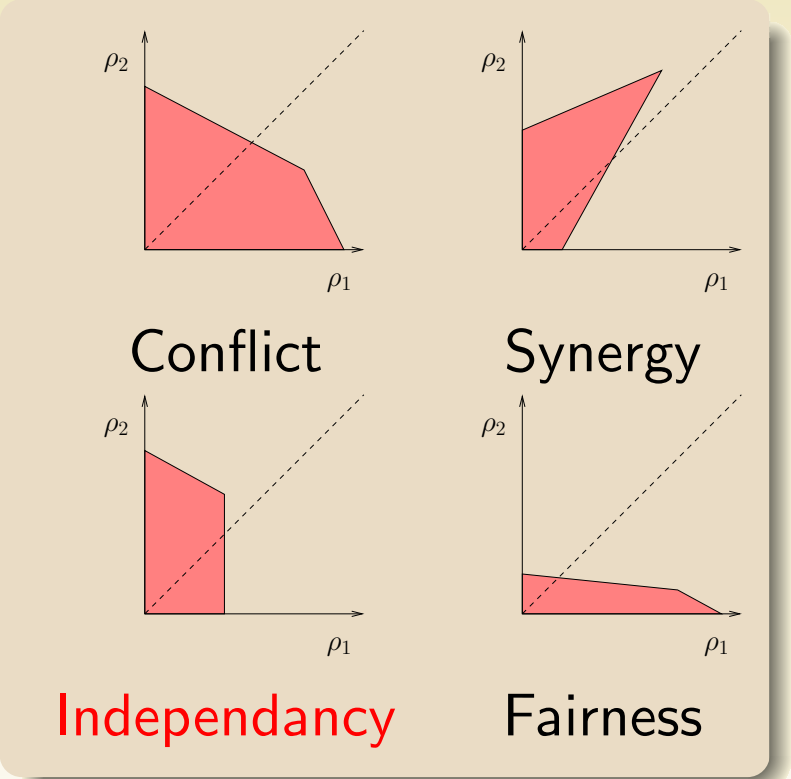




Utility Set and Fairness

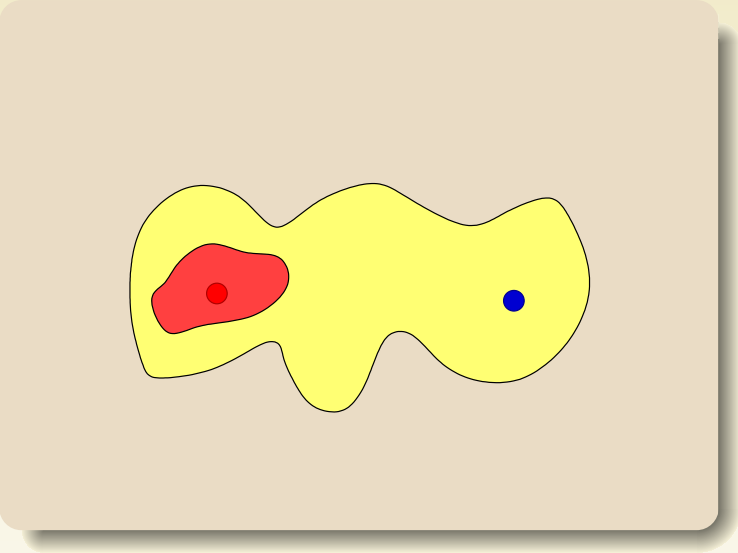
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

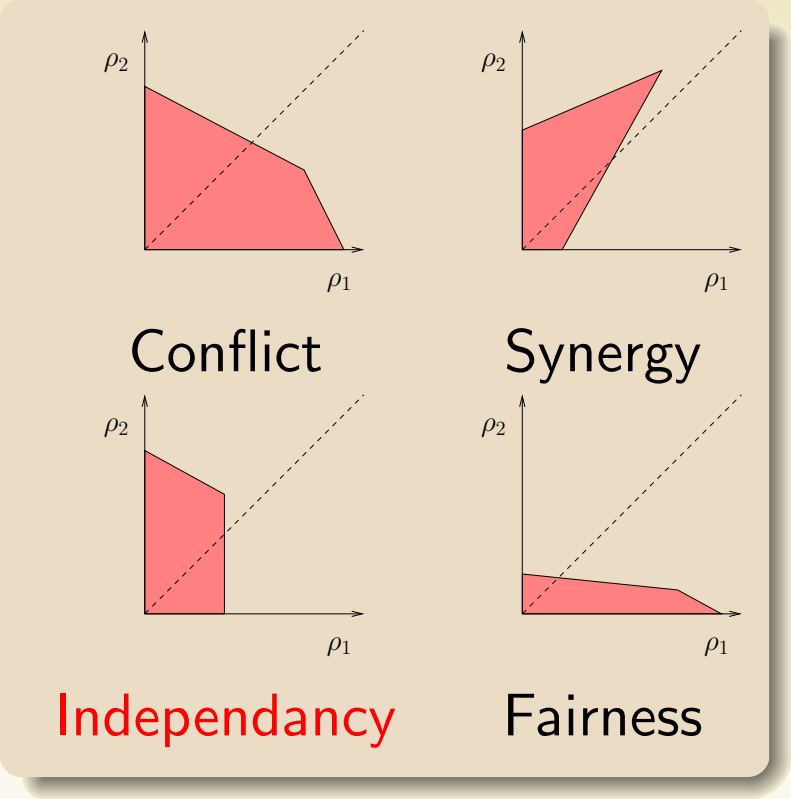




Utility Set and Fairness

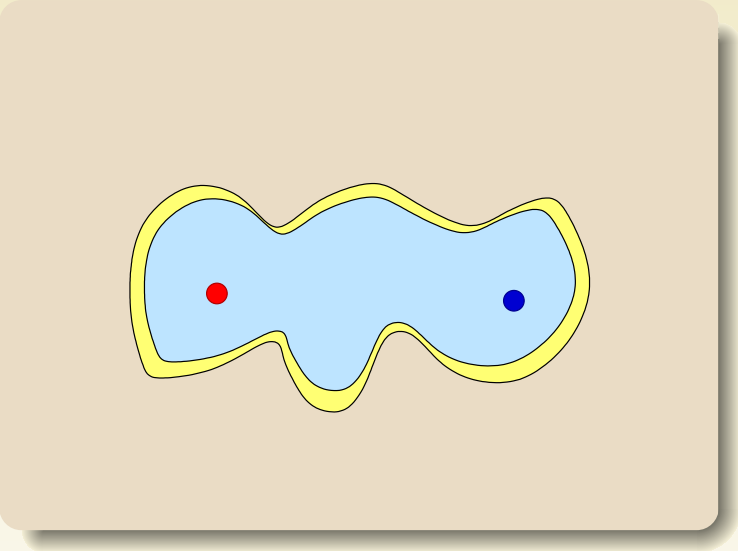
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

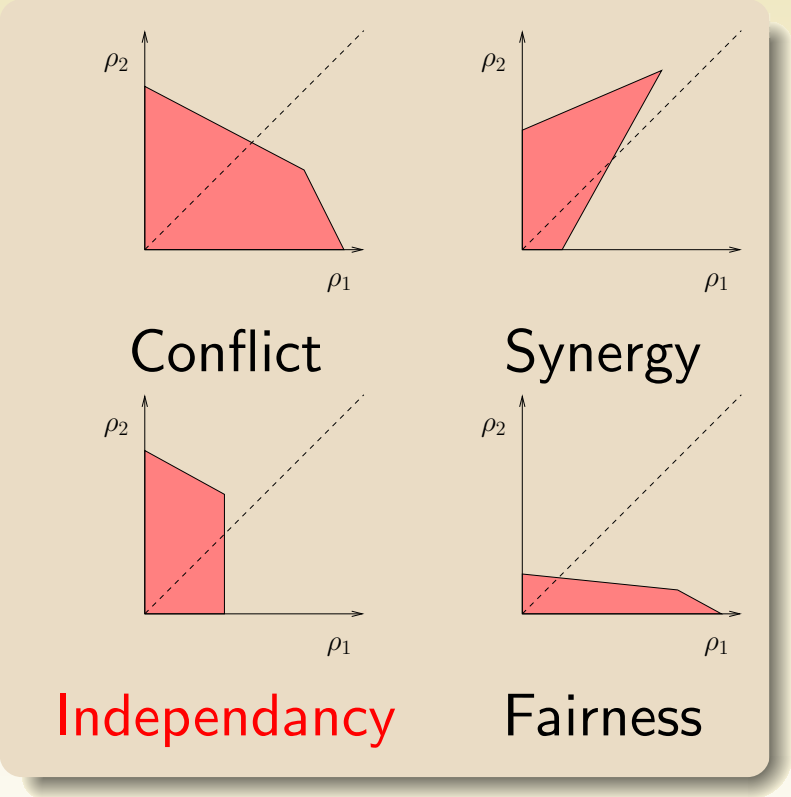




Utility Set and Fairness

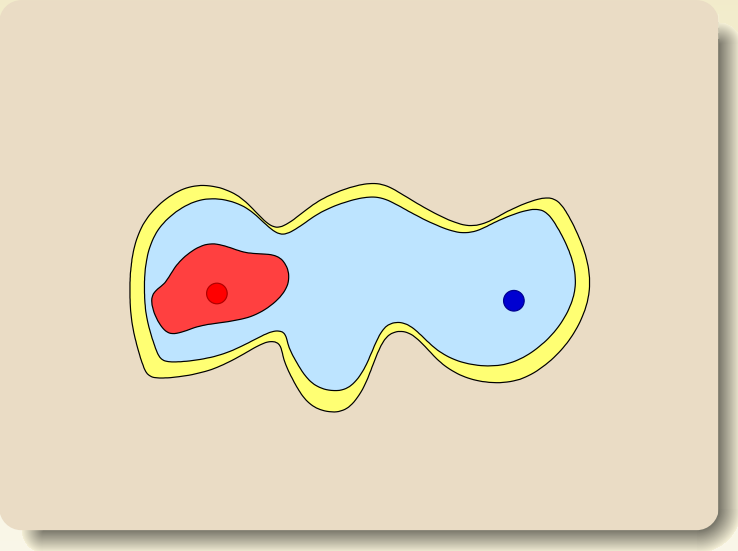
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

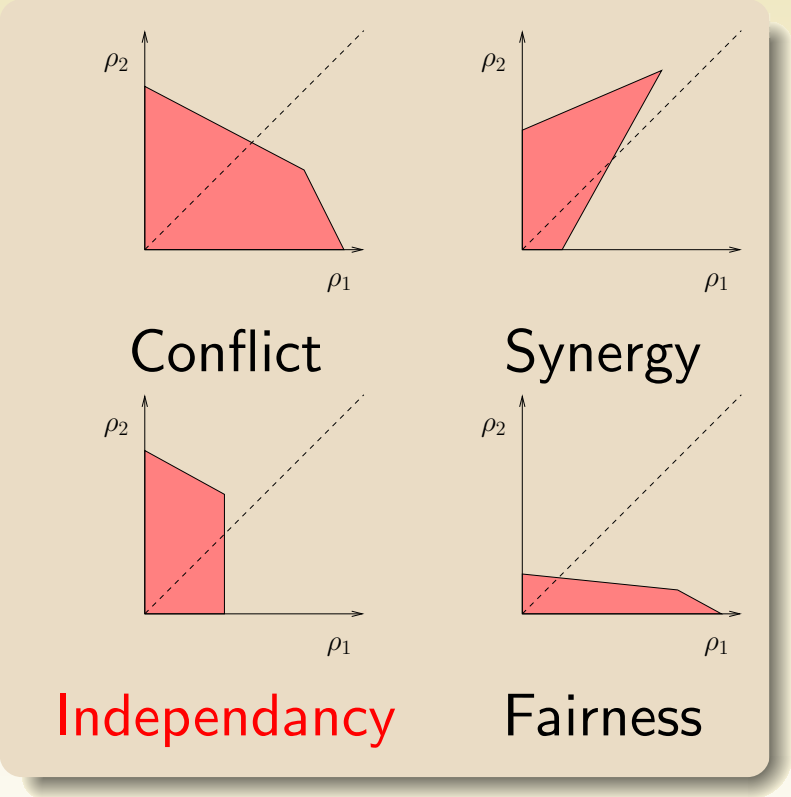




Utility Set and Fairness

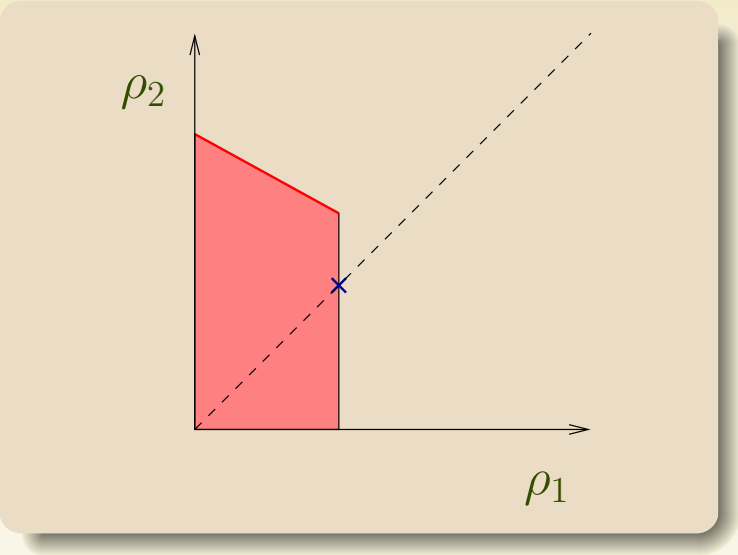
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

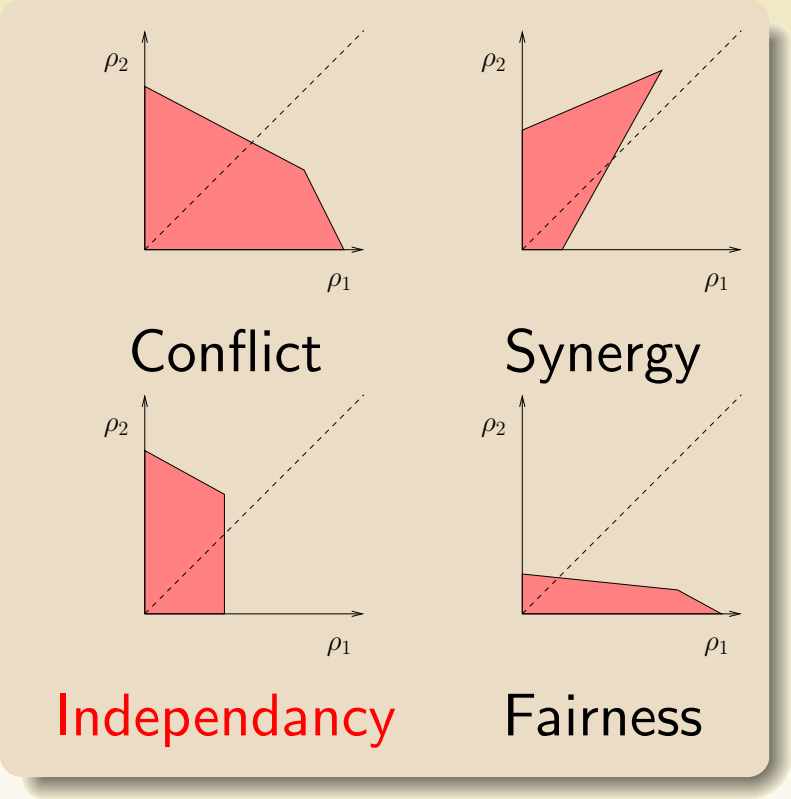




Utility Set and Fairness

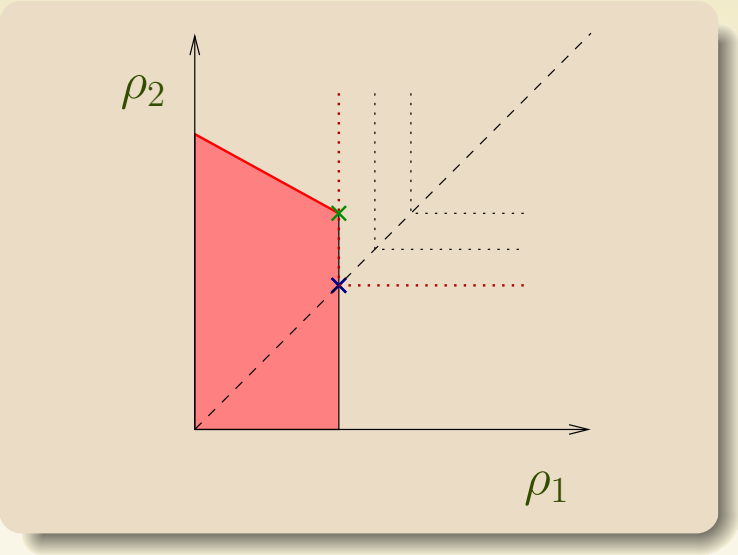
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

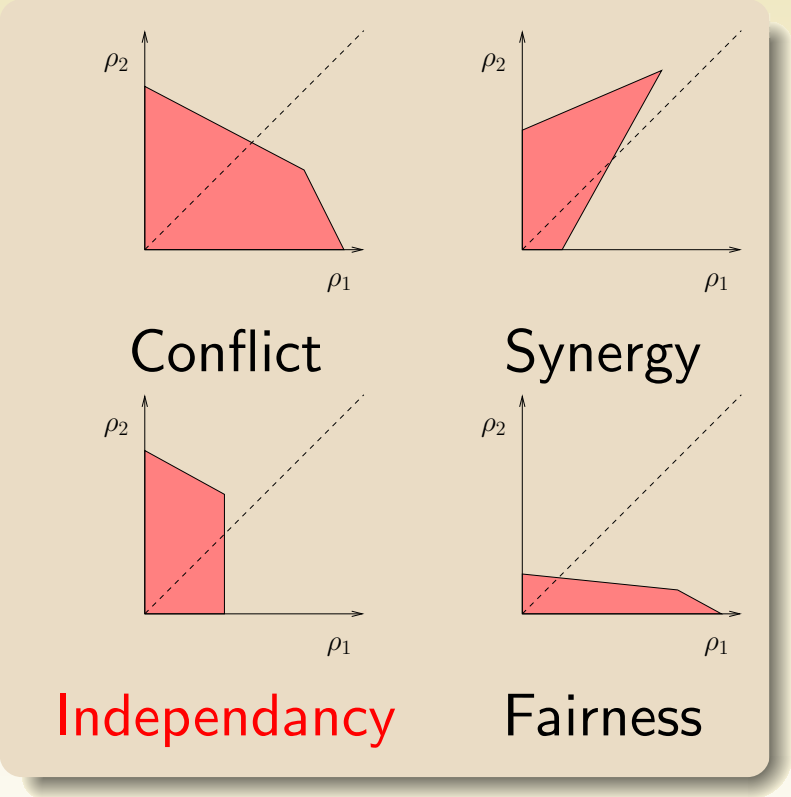




Utility Set and Fairness

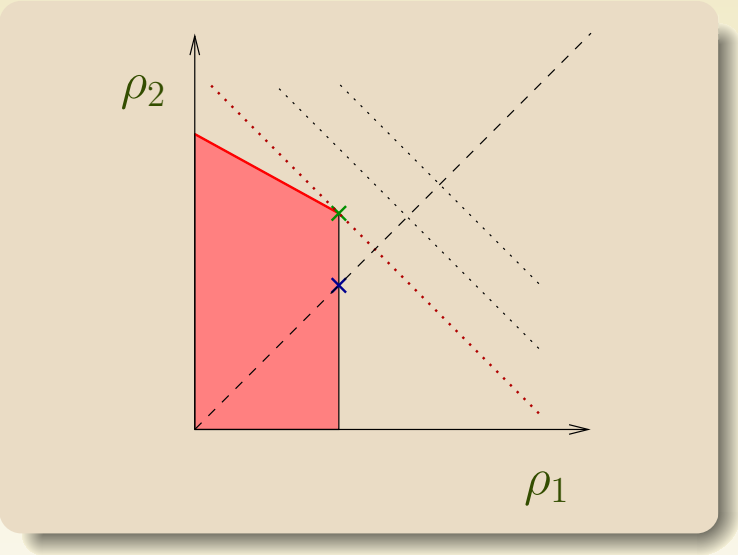
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

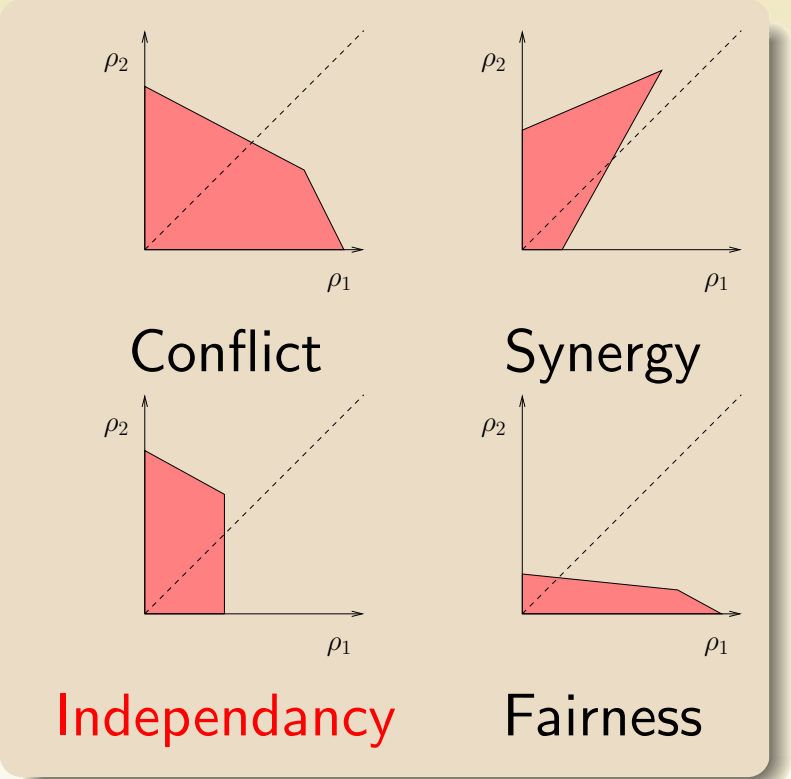




Utility Set and Fairness

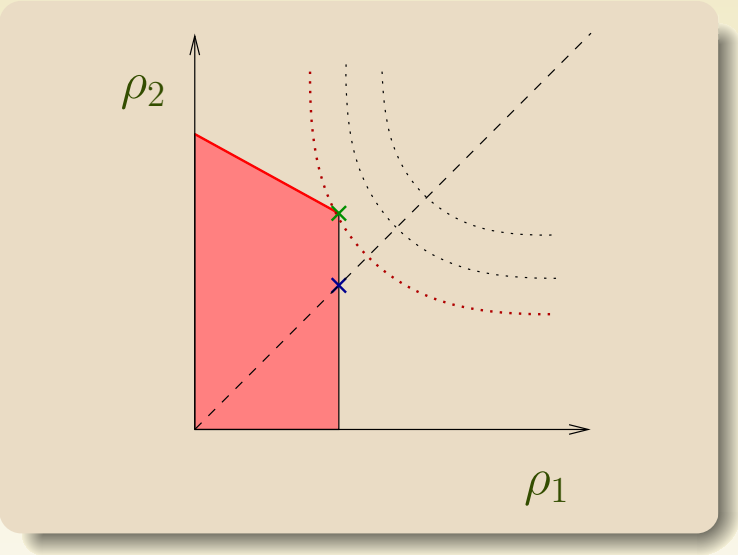
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

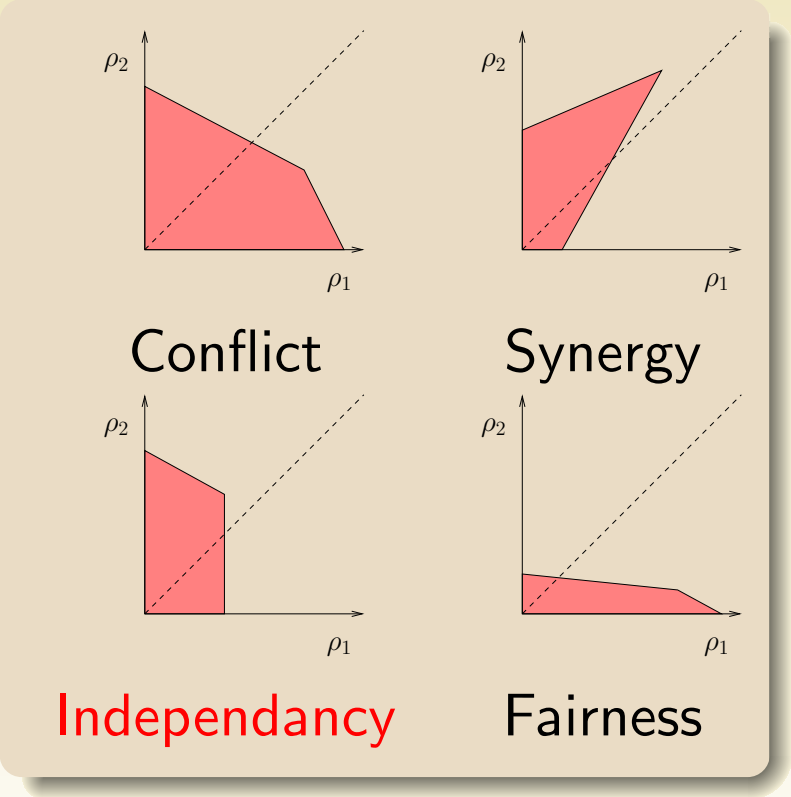




Utility Set and Fairness

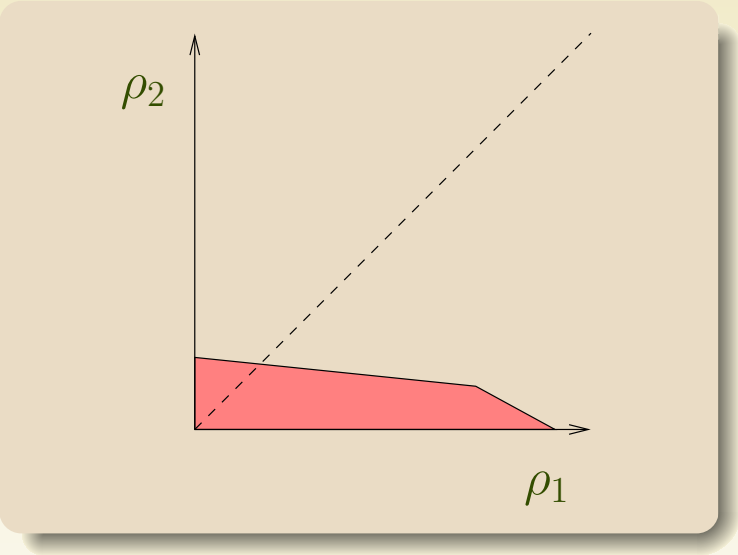
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

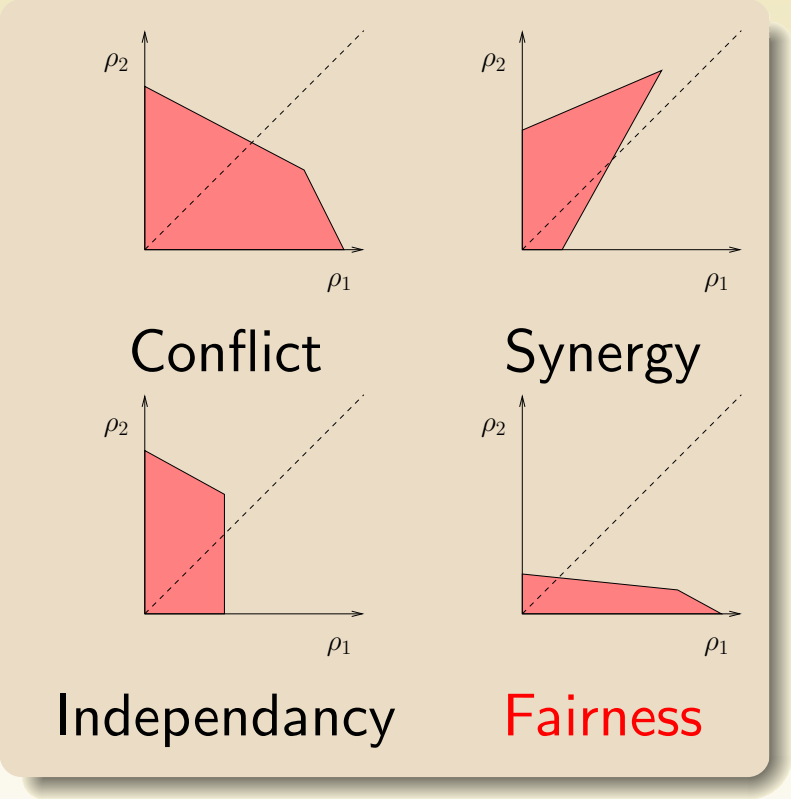




Utility Set and Fairness

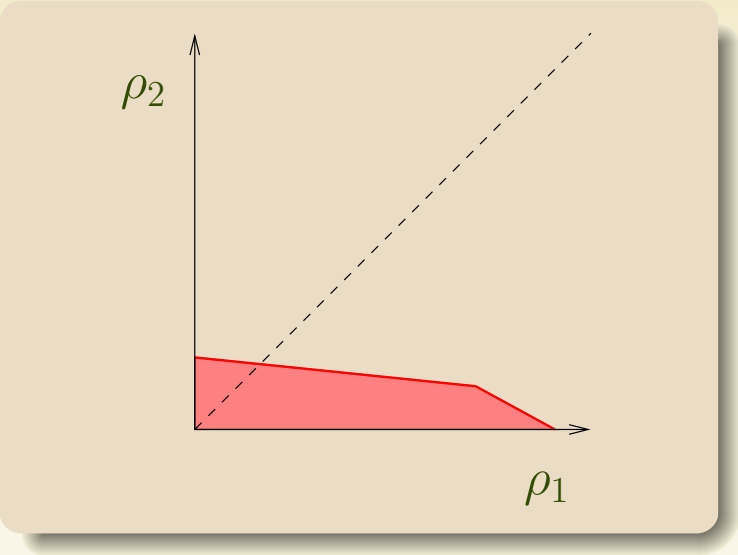
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

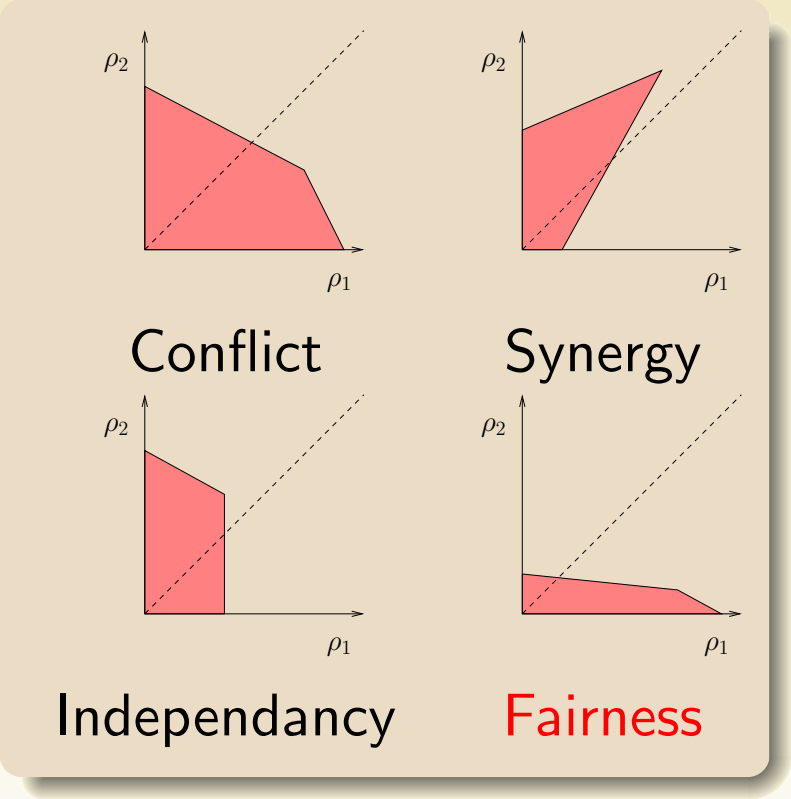




Utility Set and Fairness

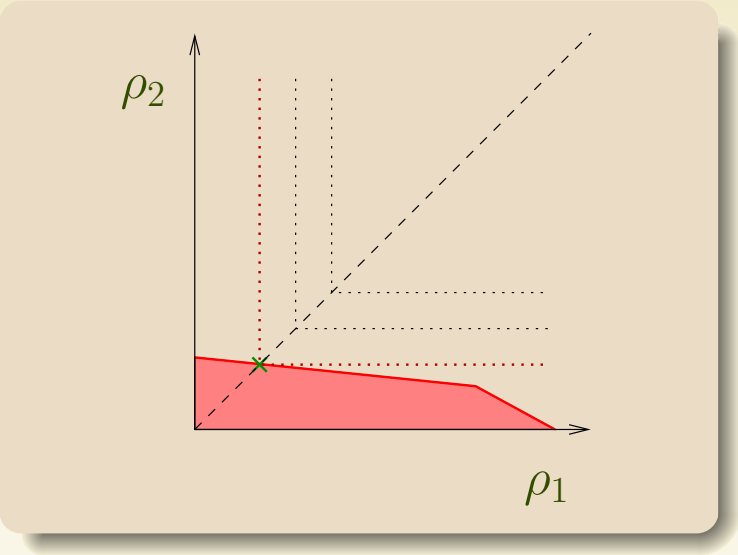
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

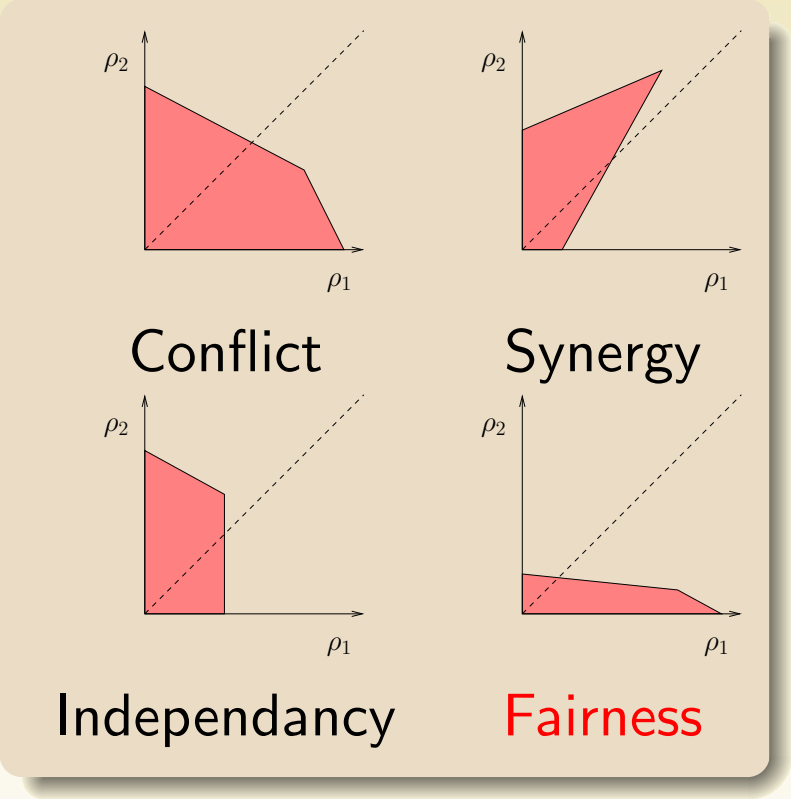




Utility Set and Fairness

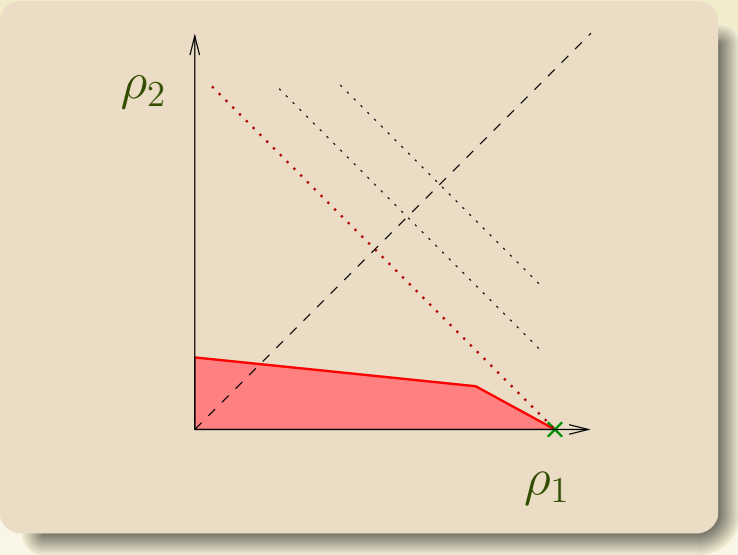
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

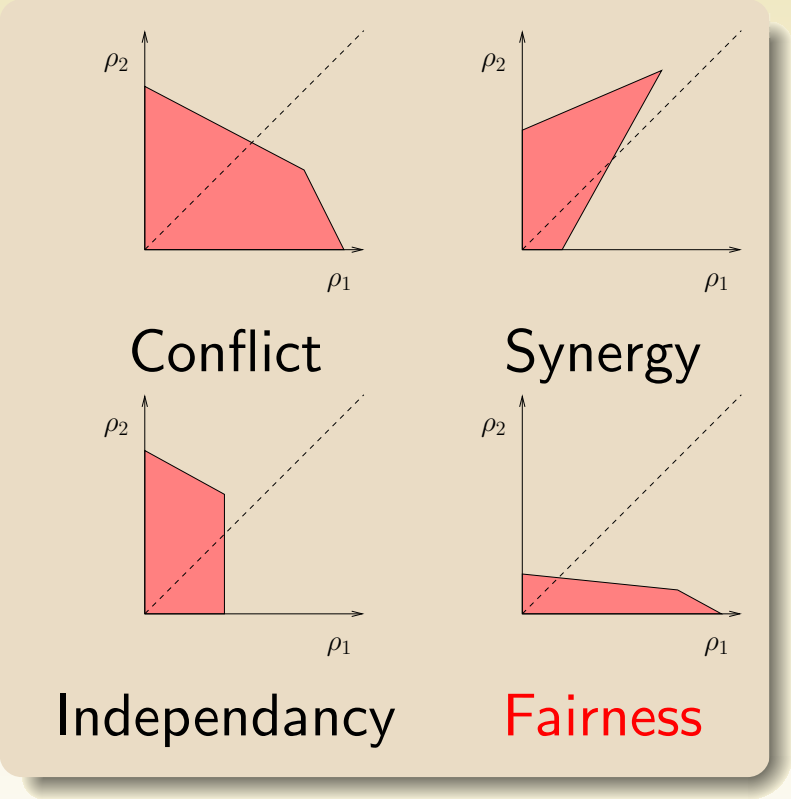




Utility Set and Fairness

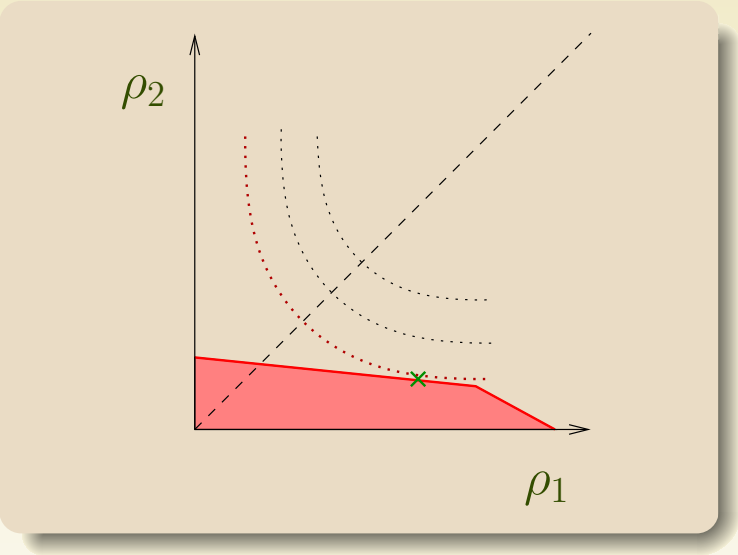
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

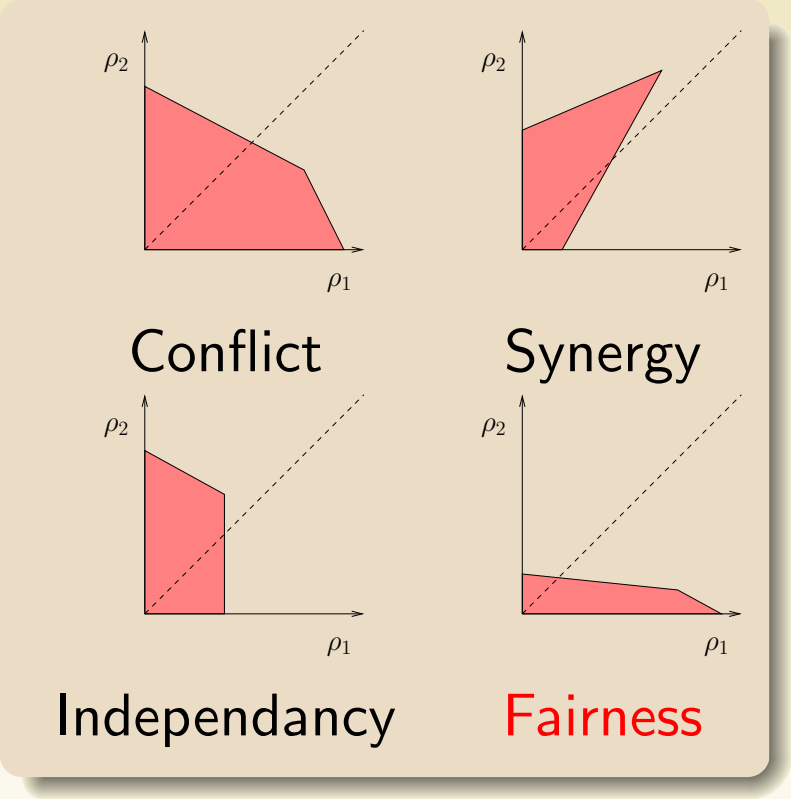




Utility Set and Fairness

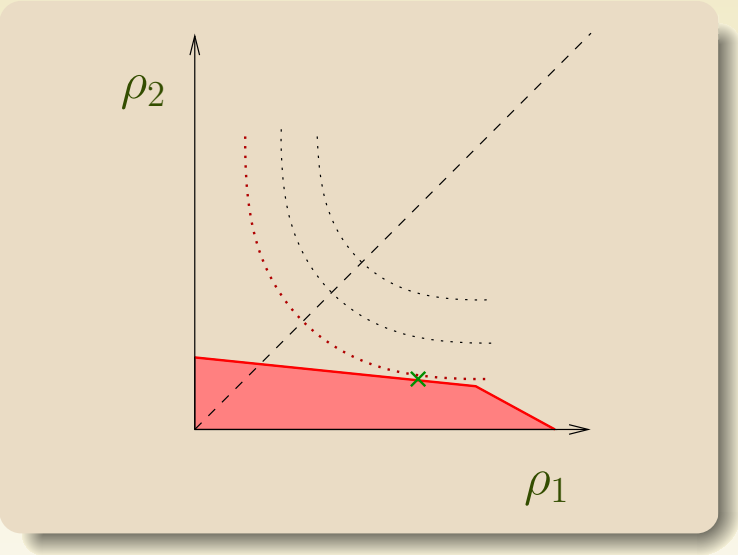
How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

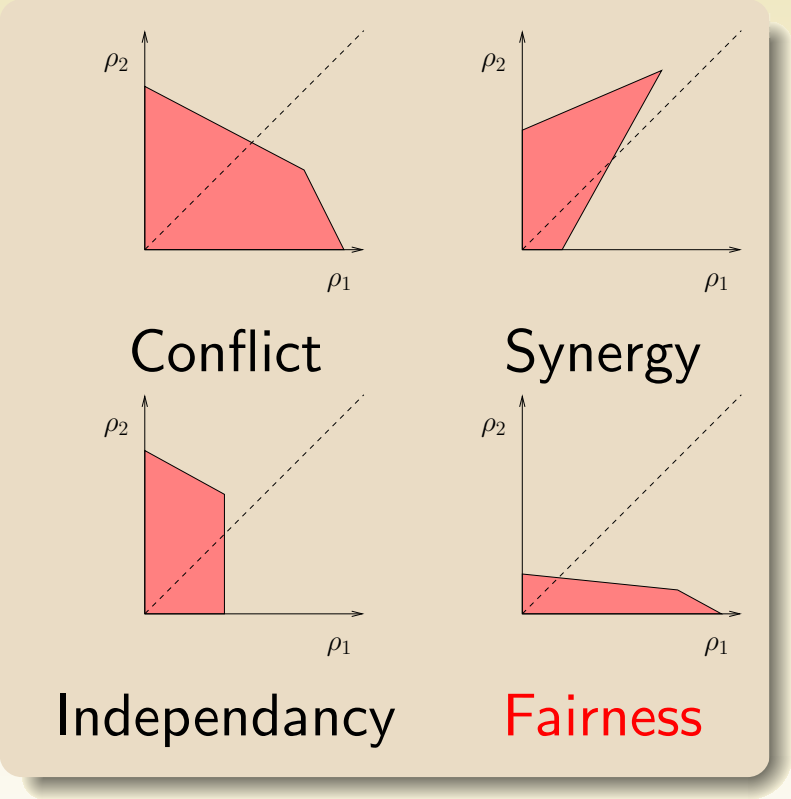




Utility Set and Fairness

How can fair be defined? Does it always mean "give the same thing to everyone"? How can efficiency be defined?

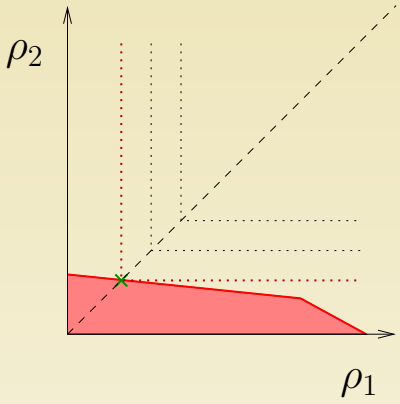




Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

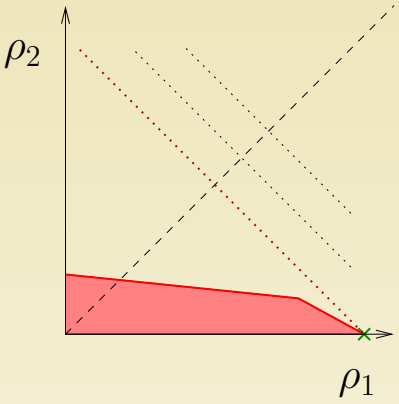
Defining Fairness for our framework

- Types d'équité:



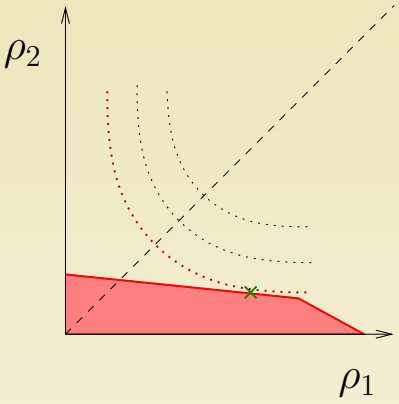
Max min

$\min_k \rho_k$



Social welfare

$\sum_k \rho_k$

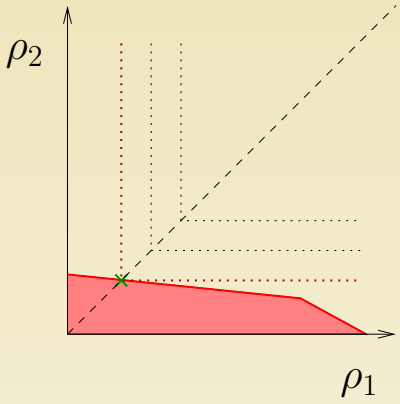


Proportional

$\prod_k \rho_k$

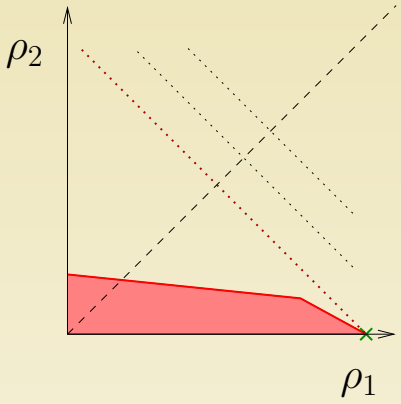
Defining Fairness for our framework

- Types d'équité:



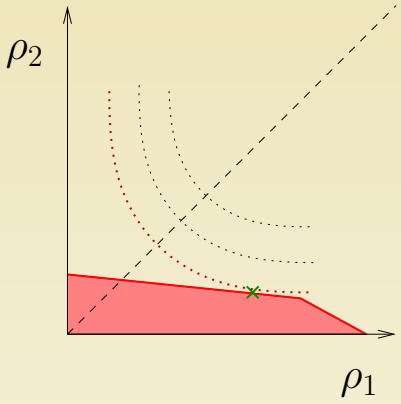
Max min

$\min_k \rho_k$



Social welfare

$\sum_k \rho_k$



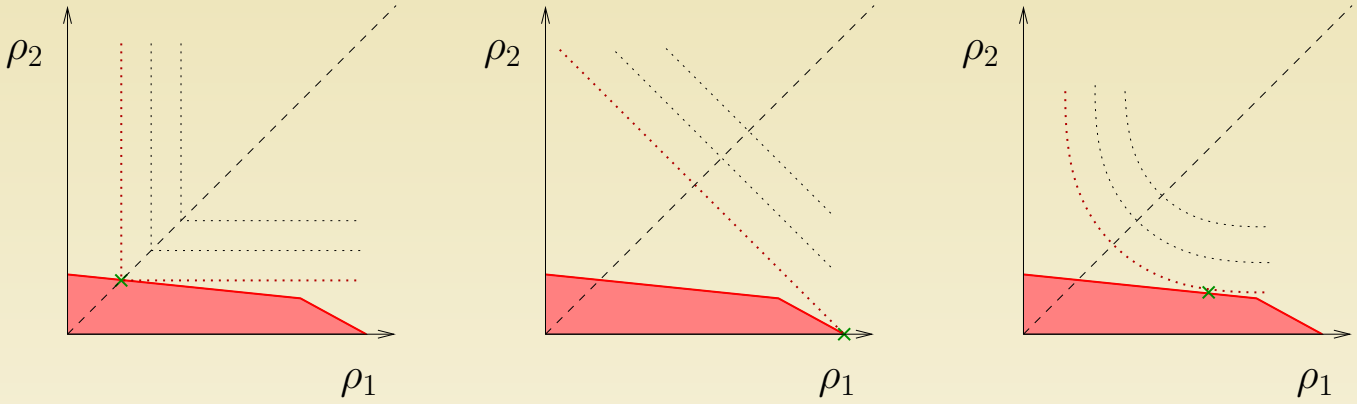
Proportional

$\prod_k \rho_k$

- A few problems with max-min fairness:

Defining Fairness for our framework

- Types d'équité:

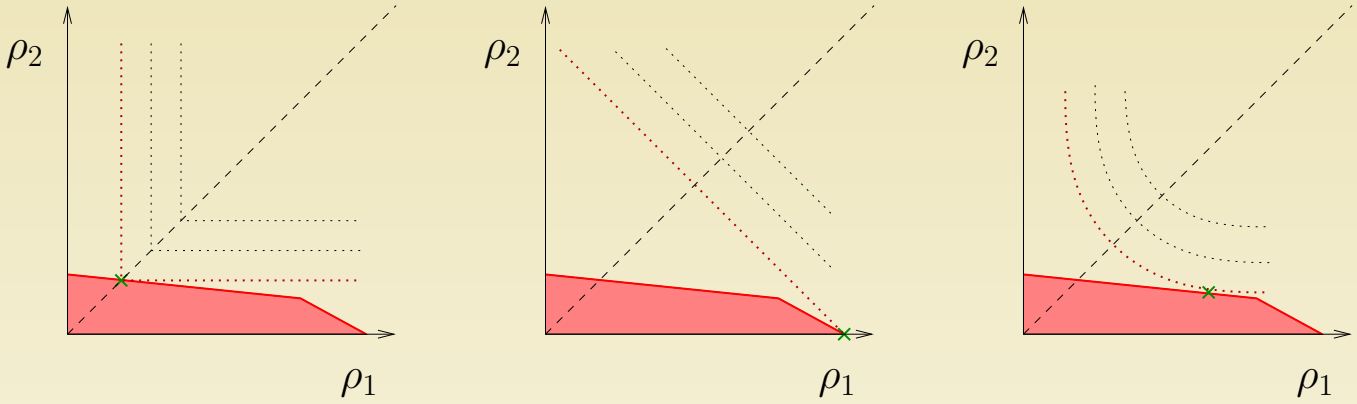


Max min	Social welfare	Proportional
$\min_k \rho_k$	$\sum_k \rho_k$	$\prod_k \rho_k$

- A few problems with max-min fairness:
  - Not "efficient" when applications are very different

Defining Fairness for our framework

- Types d'équité:

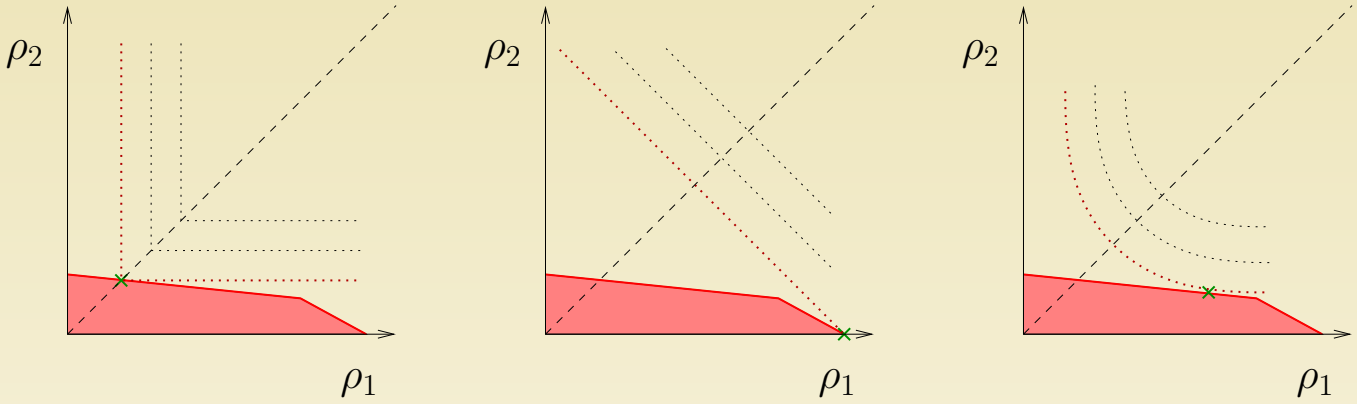


Max min	Social welfare	Proportional
$\min_k \rho_k$	$\sum_k \rho_k$	$\prod_k \rho_k$

- A few problems with max-min fairness:
  - Not "efficient" when applications are very different
  - Seems to be hard to reach on such platforms [TPDS'07]

Defining Fairness for our framework

- Types d'équité:



Max min	Social welfare	Proportional
$\min_k \rho_k$	$\sum_k \rho_k$	$\prod_k \rho_k$

- A few problems with max-min fairness:
  - Not "efficient" when applications are very different
  - Seems to be hard to reach on such platforms [TPDS'07]
- Let's try proportional fairness!

$$\text{MAXIMIZE} \left(\sum_{k \in K} \log \rho_k \right)$$

A new optimization problem

Maximize $\sum_k \log\left(\sum_n \rho_{n,k}\right)$ under the constraints:

$$\forall n, \quad \sum_k \rho_{n,k} w_k \leqslant W_n$$

$$\forall (P_i \to P_j), \quad \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \rho_{n,k} b_k \leqslant B_{i,j}$$

A new optimization problem

Maximize $\sum_k \log\left(\sum_n \rho_{n,k}\right)$ under the constraints:

$$\forall n, \quad \sum_k \rho_{n,k} w_k \leqslant W_n$$

$$\forall (P_i \to P_j), \quad \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \rho_{n,k} b_k \leqslant B_{i,j}$$

Can be solved in polynomial time e.g., with semi-definite programming. It is very centralized though.

Can we solve it in a distributed way?

Outline

Lagrangian Optimization: Basics

- Designed to solve non linear optimization problems:
  - Let $\rho \to f(\rho)$ be a function to maximize.
  - Let $(C_i(\rho) \geqslant 0)_{i \in [1..n]}$ be a set of n constraints.
  - We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\rho) \\ \forall i \in [1..n], C_i(\rho) \geqslant 0, \text{ and } \rho \geqslant 0 \end{cases}$$

Lagrangian Optimization: Basics

- Designed to solve non linear optimization problems:
  - Let $\rho \to f(\rho)$ be a function to maximize.
  - Let $(C_i(\rho) \geqslant 0)_{i \in [1..n]}$ be a set of n constraints.
  - We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\rho) \\ \forall i \in [1..n], C_i(\rho) \geqslant 0, \text{ and } \rho \geqslant 0 \end{cases}$$

- The Lagrangian function: $\mathcal{L}(\rho, \lambda) = f(\rho) - \sum_{i \in [1..n]} \lambda_i C_i(\rho)$.

Lagrangian Optimization: Basics

- Designed to solve non linear optimization problems:
  - Let $\rho \to f(\rho)$ be a function to maximize.
  - Let $(C_i(\rho) \geqslant 0)_{i\in[1..n]}$ be a set of n constraints.
  - We wish to solve:

$$(P)\begin{cases}\text{maximize } f(\rho)\\ \forall i \in [1..n], C_i(\rho) \geqslant 0, \text{ and } \rho \geqslant 0\end{cases}$$

- The Lagrangian function: $\mathcal{L}(\rho,\lambda) = f(\rho) - \sum_{i\in[1..n]} \lambda_i C_i(\rho)$.

- Under mild assumptions, there is no duality gap [Bertsekas-Tsitsiklis]:

$$\underbrace{\max_{\rho\geqslant 0}\min_{\lambda\geqslant 0}\mathcal{L}(\rho,\lambda)}_{\text{Primal problem } (P)} = \underbrace{\min_{\lambda\geqslant 0}\max_{\rho\geqslant 0}\mathcal{L}(\rho,\lambda)}_{\text{Dual problem } (D)} \stackrel{\text{def}}{=} \min_{\lambda\geqslant 0} d(\lambda)$$

Lagrangian Optimization: Basics

- D

So what?..

- Two coupled problems with simple constraints.
- $\mathcal{L}$ is concave in ρ and convex in λ

- T).
- U Tsitsiklis]:

$$\underbrace{\max_{\rho \geqslant 0} \min_{\lambda \geqslant 0} \mathcal{L}(\rho, \lambda)}_{\text{Primal problem } (P)} = \underbrace{\min_{\lambda \geqslant 0} \max_{\rho \geqslant 0} \mathcal{L}(\rho, \lambda)}_{\text{Dual problem } (D)} \stackrel{\text{def}}{=} \min_{\lambda \geqslant 0} d(\lambda)$$

Lagrangian Optimization: Basics

- D

So what?..

- Two coupled problems with simple constraints.
- $\mathcal{L}$ is concave in ρ and convex in λ
- The structure of constraints is transposed to (D) and a gradient descent algorithm is a natural way to solve these two problems.

- T).

- U Tsitsiklis]:

$$\underbrace{\max_{\rho \geqslant 0} \min_{\lambda \geqslant 0} \mathcal{L}(\rho, \lambda)}_{\text{Primal problem } (P)} = \underbrace{\min_{\lambda \geqslant 0} \max_{\rho \geqslant 0} \mathcal{L}(\rho, \lambda)}_{\text{Dual problem } (D)} \stackrel{\text{def}}{=} \min_{\lambda \geqslant 0} d(\lambda)$$

Lagrangian Optimization: Basics

- D

So what?..

- Two coupled problems with simple constraints.
- $\mathcal{L}$ is concave in ρ and convex in λ
- The structure of constraints is transposed to (D) and a gradient descent algorithm is a natural way to solve these two problems.
- This technique has been used successfully for network resource sharing [Kelly.98], TCP analysis [Low.03], flow control in multi-path network [Hang.et.al.03], ...

- T).
- U Tsitsiklis]:

$$\underbrace{\max_{\rho \geqslant 0} \min_{\lambda \geqslant 0} \mathcal{L}(\rho, \lambda)}_{\text{Primal problem } (P)} = \underbrace{\min_{\lambda \geqslant 0} \max_{\rho \geqslant 0} \mathcal{L}(\rho, \lambda)}_{\text{Dual problem } (D)} \stackrel{\text{def}}{=} \min_{\lambda \geqslant 0} d(\lambda)$$

Trying to use Lagrangian optimization

- What does the Lagrangian function look like ?

$$\mathcal{L}(\rho,\lambda,\mu) = \sum_{k\in K} \log\left(\sum_i \rho_{i,k}\right) + \sum_i \lambda_i \left(W_i - \sum_k \rho_{i,k} w_k\right)$$
$$+ \sum_{(P_i \to P_j)} \mu_{i,j} \left(B_{i,j} - \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \rho_{n,k} b_k\right)$$

Trying to use Lagrangian optimization

- What does the Lagrangian function look like ?

$$\mathcal{L}(\rho,\lambda,\mu) = \sum_{k\in K} \log\left(\sum_i \rho_{i,k}\right) + \sum_i \lambda_i \left(W_i - \sum_k \rho_{i,k} w_k\right)$$
$$+ \sum_{(P_i \to P_j)} \mu_{i,j} \left(B_{i,j} - \sum_k \sum_{\substack{n \text{ such that} \\ (P_i\to P_j)\in(P_{m(k)}\rightsquigarrow P_n)}} \rho_{n,k} b_k\right)$$

- Remember, we want to compute $\min_{\lambda,\mu\geqslant 0} \max_{\rho\geqslant 0} \mathcal{L}(\rho,\lambda,\mu)$.

$\mathcal{L}$ is concave-convex$\rightsquigarrow$ simple "alternate" gradient descent

(I'm skipping a few details here to keep it simple and just present the general idea)

Update equations: $\begin{cases} \rho_{i,k} & \leftarrow \rho_{i,k} + \gamma \frac{\partial \mathcal{L}}{\partial \rho_{i,k}} \\ \lambda_i & \leftarrow \lambda_i - \gamma \frac{\partial \mathcal{L}}{\partial \lambda_i} \\ \mu_{i,j} & \leftarrow \mu_{i,j} - \gamma \frac{\partial \mathcal{L}}{\partial \mu_{i,j}} \end{cases}$

Toward a Distributed Algorithm...

- $\rho_{i,k}$ is "private" to the agent of application k running on node i
- λ_i is attached to node i and $\mu_{i,j}$ is attached to $(P_i \to P_j)$

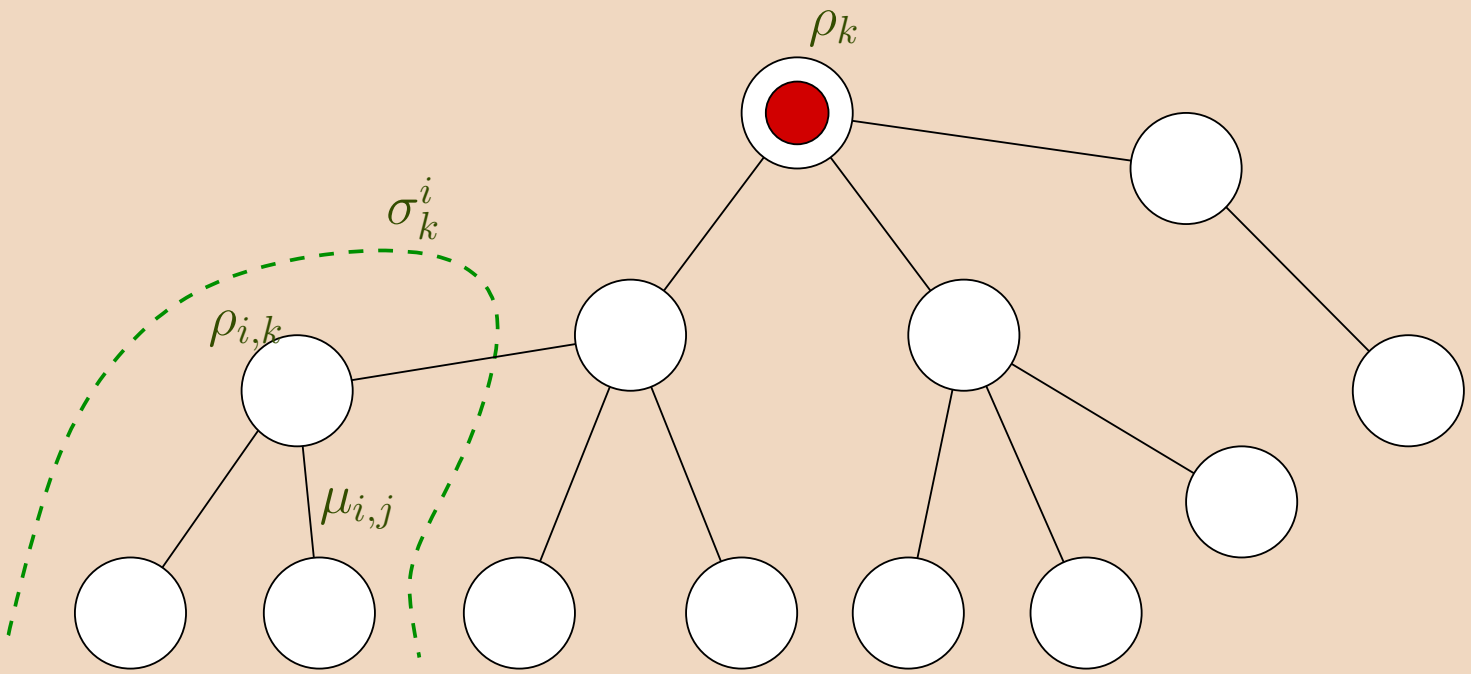
Toward a Distributed Algorithm...

- $\rho_{i,k}$ is "private" to the agent of application k running on node i
- λ_i is attached to node i and $\mu_{i,j}$ is attached to $(P_i \to P_j)$ λ_i and $\mu_{i,j}$ are called shadow variables or shadow prices. They can naturally thought of as the **price to pay to use the corresponding resource**.
- A gradient descent algorithm on the primal-dual problem can thus be seen as a bargain between applications and resources.

Toward a Distributed Algorithm...

- ρ
- λ
  a
  c
  r
- A
  t
- W
  i
  t

Hierarchical deployment

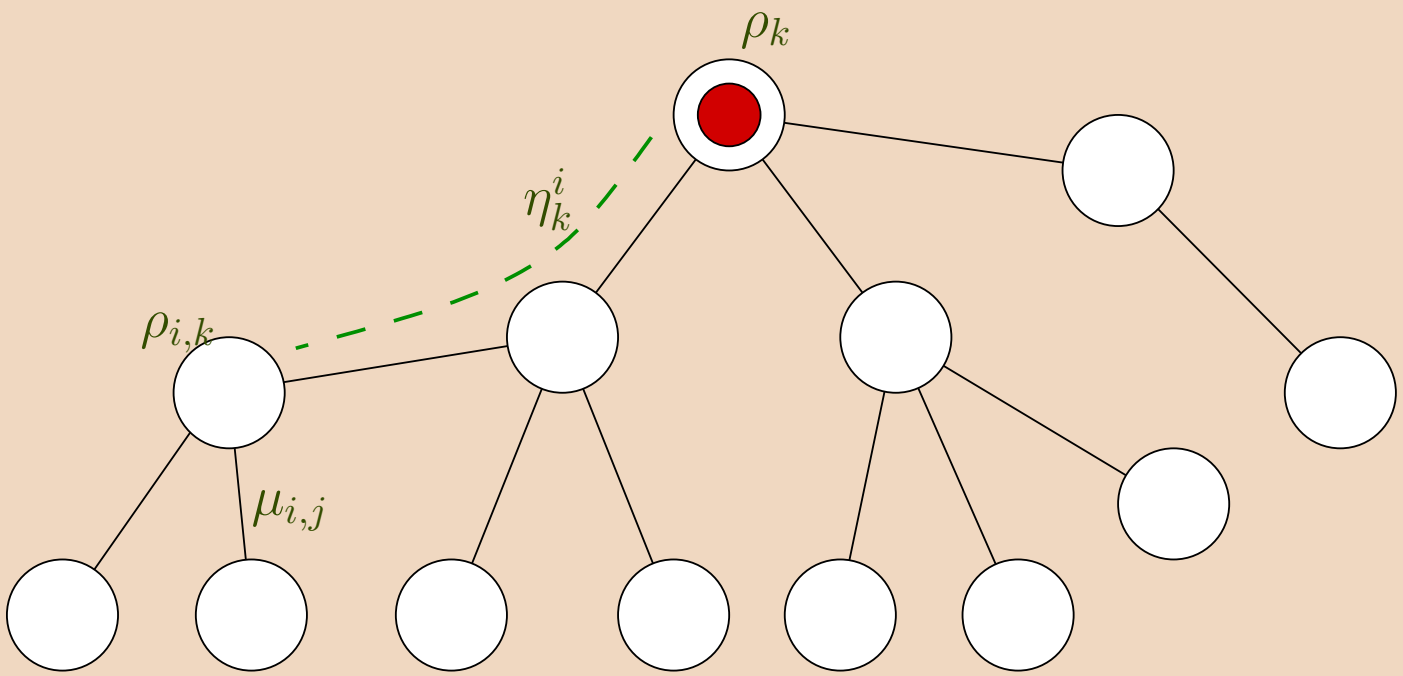


$$\begin{cases} \sigma_k^n = \displaystyle\sum_{p \text{ such that } n \in (P_{m(k)} \rightsquigarrow P_p)} \rho_{p,k} & \left\{\begin{array}{l}\text{aggregate throughput}\\ \text{of a subtree.}\end{array}\right. \\ \eta_k^n = \displaystyle\sum_{(P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)} \mu_{i,j} & \left\{\begin{array}{l}\text{aggregate communication}\\ \text{price}\end{array}\right. \end{cases}$$

Toward a Distributed Algorithm...

- ρ
- λ
  a
  c
  r
- A
  t
- W
  i
  t

Hierarchical deployment



$$\begin{cases} \sigma_k^n = \displaystyle\sum_{p \text{ such that } n \in (P_{m(k)} \rightsquigarrow P_p)} \rho_{p,k} & \left\{\begin{array}{l}\text{aggregate throughput}\\ \text{of a subtree.}\end{array}\right. \\ \eta_k^n = \displaystyle\sum_{(P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)} \mu_{i,j} & \left\{\begin{array}{l}\text{aggregate communication}\\ \text{price}\end{array}\right. \end{cases}$$

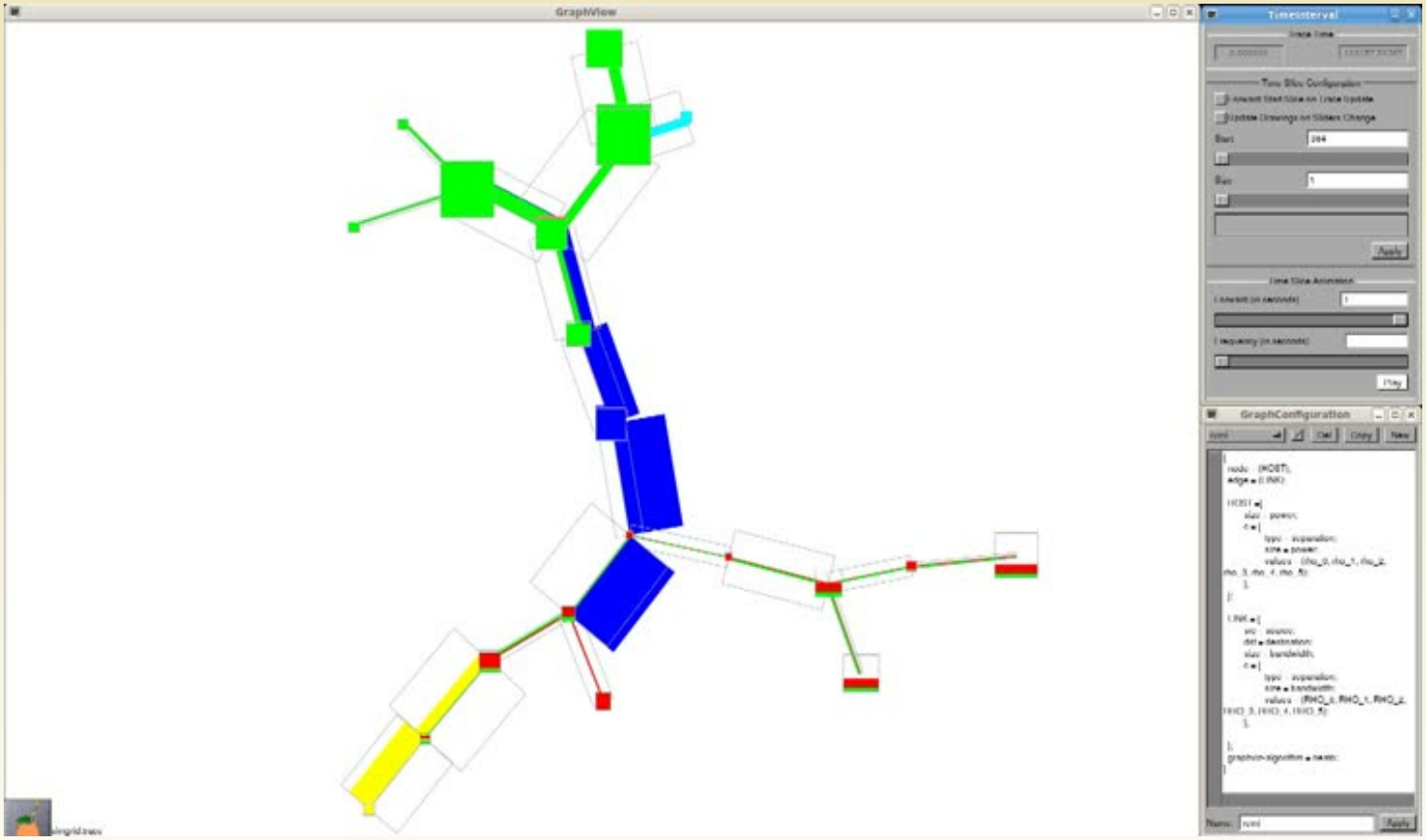
Toward a Distributed Algorithm...

Prices and rates can thus be propagated and aggregated to perform the following updates:

$$
\begin{aligned}
p_k^i(t+1) &\leftarrow b_k \eta_k^i(t) + w_k \lambda_i(t) \\
\rho_k(t+1) &\leftarrow \sigma_k^{m(k)}(t+1) \\
\rho_{i,k}(t+1) &\leftarrow \left[\rho_{i,k}(t) + \gamma_\rho (U_k'(\rho_k(t)) - p_k^i(t))\right]^+ \\
\lambda_i(t+1) &\leftarrow \left[\lambda_i(t) + \gamma_\lambda \left(\sum_k w_k \rho_{i,k} - W_i\right)\right]^+ \\
\mu_{i,j}(t+1) &\leftarrow \left[\mu_{i,j}(t) + \gamma_\mu \left(\sum_k b_k \sigma_k^i - B_{i,j}\right)\right]^+
\end{aligned}
$$

- This algorithm is fully distributed and converges to the optimal solution provided a good choice of γ_ρ, γ_λ and γ_μ is done.
- This algorithm should seamlessly adapts to application/node arrival and to load variations.

Illustration of convergence on a toy platform



Wrap up

There are three main steps to come up with such an algorithm:

1. **Modeling:** concave non-linear maximization problem
2. **Partial derivatives:** differentiate $\mathcal{L}$
3. **Algorithm design:** ascent on primal and descent on dual exploiting the structure of derivatives

Wrap up

There are three main steps to come up with such an algorithm:

1. **Modeling:** concave non-linear maximization problem
2. **Partial derivatives:** differentiate $\mathcal{L}$
3. **Algorithm design:** ascent on primal and descent on dual exploiting the structure of derivatives

The key ingredients are:

- the separability of the objective function
- the structure of the constraints

Wrap up

There are three main steps to come up with such an algorithm:

1. **Modeling:** concave non-linear maximization problem
2. **Partial derivatives:** differentiate $\mathcal{L}$
3. **Algorithm design:** ascent on primal and descent on dual exploiting the structure of derivatives

The key ingredients are:

- the separability of the objective function
- the structure of the constraints

"Technical" issues for convergence:

- Beware of divisions by 0
- Strict convexity of the objective function

Wrap up

There are three main steps to come up with such an algorithm:

1. **Modeling:** concave non-linear maximization problem
2. **Partial derivatives:** differentiate $\mathcal{L}$
3. **Algorithm design:** ascent on primal and descent on dual exploiting the structure of derivatives

The key ingredients are:

- the separability of the objective function
- the structure of the constraints

"Technical" issues for convergence:

- Beware of divisions by 0
- Strict convexity of the objective function ($\rightsquigarrow$proximal variables $\widetilde{\rho}$)

$$\max_{\widetilde{\rho}\geqslant 0} \max_{\rho\geqslant 0} \min_{\lambda\geqslant 0} L(\widetilde{\rho}, \rho, \lambda),$$

- Nested structure needs to be broken in practice!!!

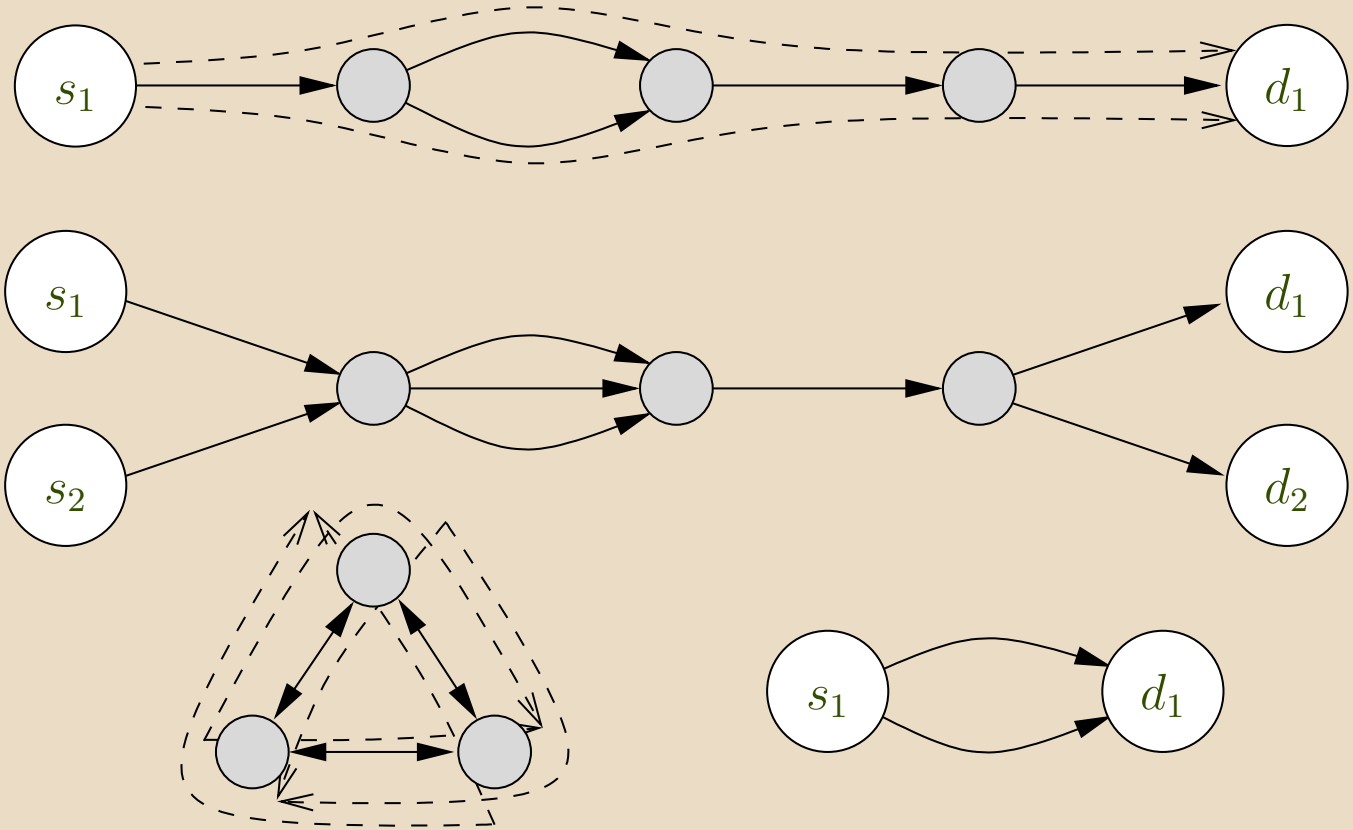
And in practice ?

In 2006, we stumbled on [Hang.et.al.03]: "Optimal Flow Control and Routing in Multi-path Networks", where we learnt about these techniques

And in practice ?

In 2006, we stumbled on [Hang.et.al.03]: "Optimal Flow Control and Routing in Multi-path Networks", where we learnt about these techniques

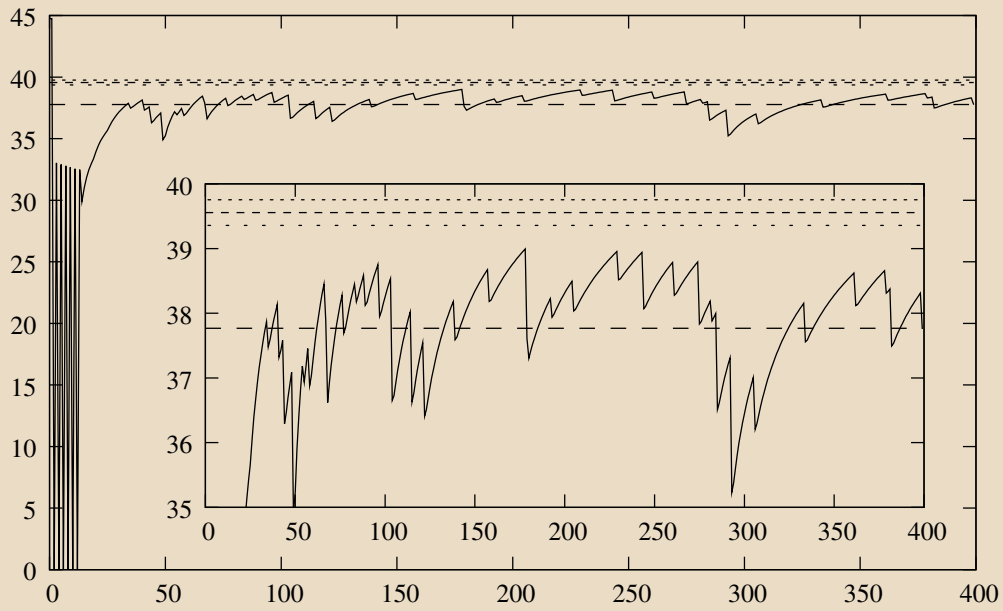
Effectiveness of the approach was illustrated on a few simple cases



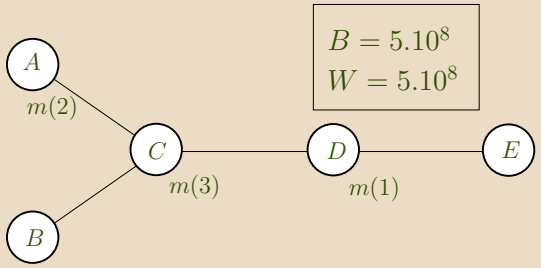
And in practice ?

In 2006, we stumbled on [Hang.et.al.03]: "Optimal Flow Control and Routing in Multi-path Networks", where we learnt about these techniques

Using the same formulas did not work out well...



- Over sensitive to step sizes
- Never converges even on toy platforms



Outline

Experimental Setting

- SimGrid based simulation
- Check correctness with semi-definite programming
- Adjusting 4 different steps at a time seems tricky:
  - use designed experiments and Analysis of Variance to assess the effectiveness of step modification
  - Use the same 30 platforms for each step configuration and distinguish platform variability from algorithm variability

Experimental Setting

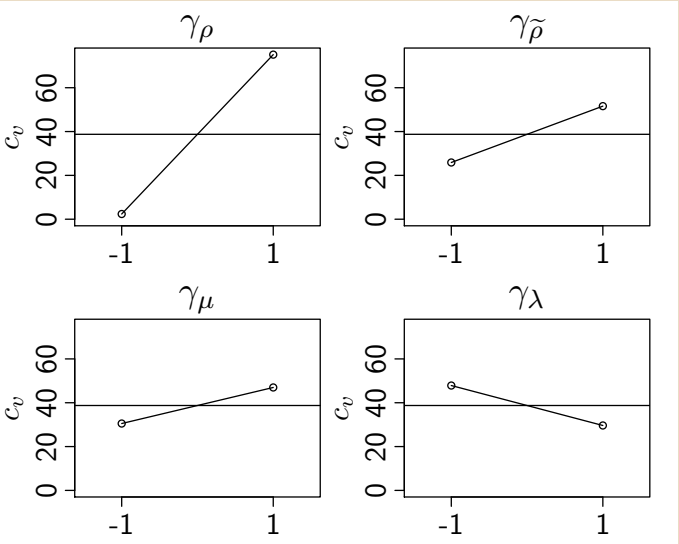
A typical experiment

	γ_ρ	$\gamma_{\widetilde{\rho}}$	γ_μ	γ_λ
low (-1)	0.001	0.001	1.00e-15	1.00e-15
high (1)	0.1	0.1	1.00e-13	1.00e-13

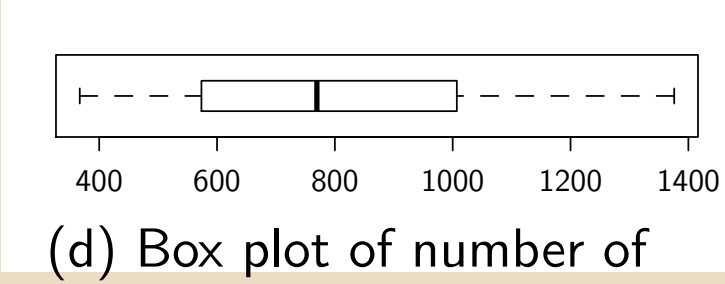
(a) Parameters for factorial design

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
ρ	1	634854.4356	634854.4356	75.1771	0.0000	***
$\widetilde{\rho}$	1	79293.1822	79293.1822	9.3896	0.0023	**
λ	1	39882.0712	39882.0712	4.7227	0.0303	*
μ	1	32497.4344	32497.4344	3.8482	0.0504	.
platform	29	470172.3060	16212.8381	1.9199	0.0032	**
$\rho : \widetilde{\rho}$	1	67441.2012	67441.2012	7.9861	0.0049	**
$\rho : \lambda$	1	27584.2533	27584.2533	3.2664	0.0714	.
$\widetilde{\rho} : \lambda$	1	330.0890	330.0890	0.0391	0.8434	
$\rho : \mu$	1	38450.8702	38450.8702	4.5532	0.0334	*
$\widetilde{\rho} : \mu$	1	31651.5862	31651.5862	3.7481	0.0535	.
$\lambda : \mu$	1	14136.1931	14136.1931	1.6740	0.1964	
$\rho : \widetilde{\rho} : \lambda$	1	1489.7248	1489.7248	0.1764	0.6747	
$\rho : \widetilde{\rho} : \mu$	1	35856.2101	35856.2101	4.2460	0.0399	*
$\rho : \lambda : \mu$	1	16345.9826	16345.9826	1.9356	0.1649	
$\widetilde{\rho} : \lambda : \mu$	1	16.1140	16.1140	0.0019	0.9652	
$\rho : \widetilde{\rho} : \lambda : \mu$	1	158.4895	158.4895	0.0188	0.8911	
Residuals	435	3673480.0717	8444.7818			

(c) ANOVA results: * means that parameter is significant



(b) Main effects plot



(d) Box plot of number of iterations until convergence for best parameter set

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature
  1. Unstable equilibrium:

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature
  1. Unstable equilibrium:

$\Delta\lambda_n$ Primal

$\Delta\rho_{n,k} = -\alpha.\Delta\lambda_n$ Dual

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature
  1. Unstable equilibrium:
     Rescale so that $\alpha\beta < 1$!!!

$\Delta\lambda_n = -\alpha\beta.\Delta\lambda_n$ Primal

$\Delta\rho_{n,k} = -\alpha.\Delta\lambda_n$ Dual

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature
  1. Unstable equilibrium:
     Rescale so that $\alpha\beta < 1$!!!
  2. Slow convergence: use **quasi-Newton** scheme

$\Delta\lambda_n = -\alpha\beta.\Delta\lambda_n$ Primal

$\Delta\rho_{n,k} = -\alpha.\Delta\lambda_n$ Dual

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature
  1. Unstable equilibrium:
     Rescale so that $\alpha\beta < 1$!!!
  2. Slow convergence: use **quasi-Newton** scheme
  3. Division by small values and discontinuities:
     $[x(t)]^+ = \max(0, x(t))$.

$\Delta\lambda_n = -\alpha\beta.\Delta\lambda_n$ Primal

$\Delta\rho_{n,k} = -\alpha.\Delta\lambda_n$ Dual

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature
  1. Unstable equilibrium:
     Rescale so that $\alpha\beta < 1$!!!

     $\Delta\lambda_n = -\alpha\beta.\Delta\lambda_n$ Primal

     $\Delta\rho_{n,k} = -\alpha.\Delta\lambda_n$ Dual
  2. Slow convergence: use **quasi-Newton** scheme
  3. Division by small values and discontinuities:
     $\cancel{[x(t)]^+ = \max(0, x(t))}$. Use **exponential decrease** instead.

Conclusion of the investigation

Here is what we found out:

- The algorithm works great when using identical applications, even with 100 nodes
- It is a complete failure otherwise
- This unstability has several causes but not the ones usually mentionned in the litterature
  1. Unstable equilibrium:
     Rescale so that $\alpha\beta < 1$!!!

     $\Delta\lambda_n = -\alpha\beta.\Delta\lambda_n$ Primal

     $\Delta\rho_{n,k} = -\alpha.\Delta\lambda_n$ Dual
  2. Slow convergence: use **quasi-Newton** scheme
  3. Division by small values and discontinuities:
     $\cancel{[x(t)]^+ = \max(0, x(t))}$. Use **exponential decrease** instead.
- Combining these three techniques provides a distributed algorithm that converges even for platforms with 500 nodes.

Open remarks

- Similitude between fair steady-state scheduling and flow control in multi-path networks motivated the Lagrangian approach.

  All the convergence issues would have been overlooked if we had been less stubborn

Open remarks

- Similitude between fair steady-state scheduling and flow control in multi-path networks motivated the Lagrangian approach.

  All the convergence issues would have been overlooked if we had been less stubborn
- **Theory vs. "Practice"**

  Theory difficulty lies in non strict convexity.
  Practice scaling and borders are the most important issues.

  Using designed experiments we can find "robust" step sizes

Open remarks

- Similitude between fair steady-state scheduling and flow control in multi-path networks motivated the Lagrangian approach.

  All the convergence issues would have been overlooked if we had been less stubborn
- **Theory vs. "Practice"**

  Theory difficulty lies in non strict convexity.
  Practice scaling and borders are the most important issues.

  Using designed experiments we can find "robust" step sizes
- The exponential decay reminds of a barrier function $\rightsquigarrow$ interior point methods ? I still lack perspective on these problems but such optimization techniques are still ongoing research.

Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, Loris Marchal, and Yves Robert.
Centralized versus distributed schedulers multiple bag-of-tasks applications.
IEEE Trans. Parallel Distributed Systems, 2007.

D. Braess.
Über ein paradoxien aus der verkehrsplanung.
Unternehmensforschung, 12:258–68, 1968.

D. P. Bertsekas and J. N. Tsitsiklis.
Parallel and Distributed Computation: Numerical Methods.
Prentice-Hall, 1989.

Frank Kelly, Aman Maulloo, and David Tan.
Rate control in communication networks: shadow prices, proportional fairness and stability.
Journal of the Operational Research Society, 49:237–252, 1998.

E. Koutsoupias and C. Papadimitriou.
Worst-case equilibria.

In *STACS*, 1998.

Steven Low.
A duality model of TCP and queue management algorithms.
IEEE/ACM Transactions on Networking, 11(4):525–536, 2003.

Arnaud Legrand and Corinne Touati.
Non-cooperative scheduling of multiple bag-of-task appplications.
In *Proceedings of the 25th Conference on Computer Communications (INFOCOM'07)*, Alaska, USA, May 2007.

John F. Nash.
Noncooperative games.
Annal of Mathematics, 54:286–295, 1951.

C. H. Papadimitriou and M. Yannakakis.
On the approximability of trade-offs and optimal access of web sources.

In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 86, Washington, DC, USA, 2000. IEEE Computer Society.

Wei-Hua Wang, Marimuthu Palaniswami, and Steven Low. Optimal flow control and routing in multi-path networks. *Performance Evaluation*, 52:119–132, 2003.