Competitive Strategies for Online Cloud Resource Allocation with Discounts

The 2-Dimensional Parking Permit Problem

Xinhui Hu, **Arne Ludwig**, Andrea Richa, Stefan Schmid

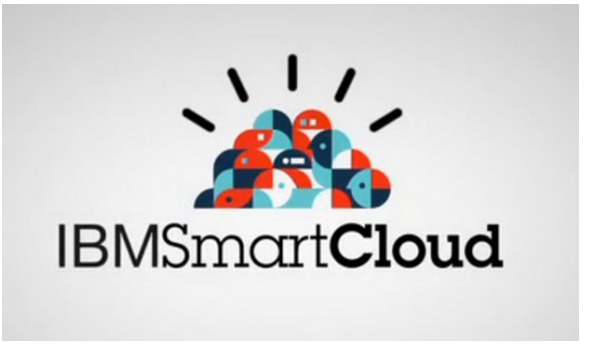
Motivation

- 30% CAGR cloud 2013-2018

  → Plethora of cloud providers

Google Cloud Platform

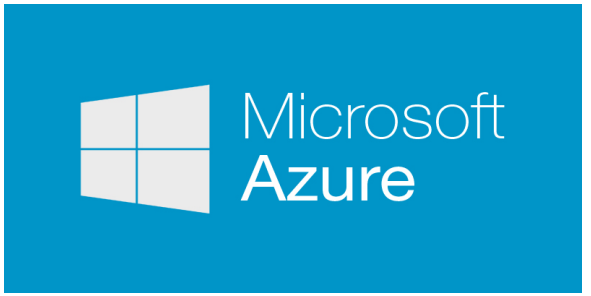
rackspace®
the open cloud company



IBMSmartCloud

salesforce

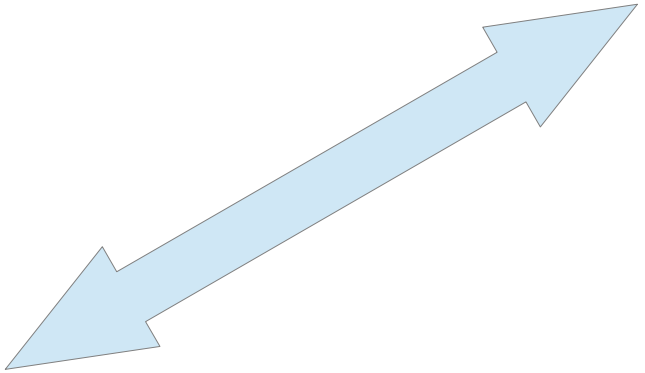
amazon
web services™



Microsoft
Azure

Motivation

- Virtualization enables new business models
  - Dynamic leasing (startups)
  - Cloud bursting



salesforce

Microsoft Azure

IBMSmartCloud

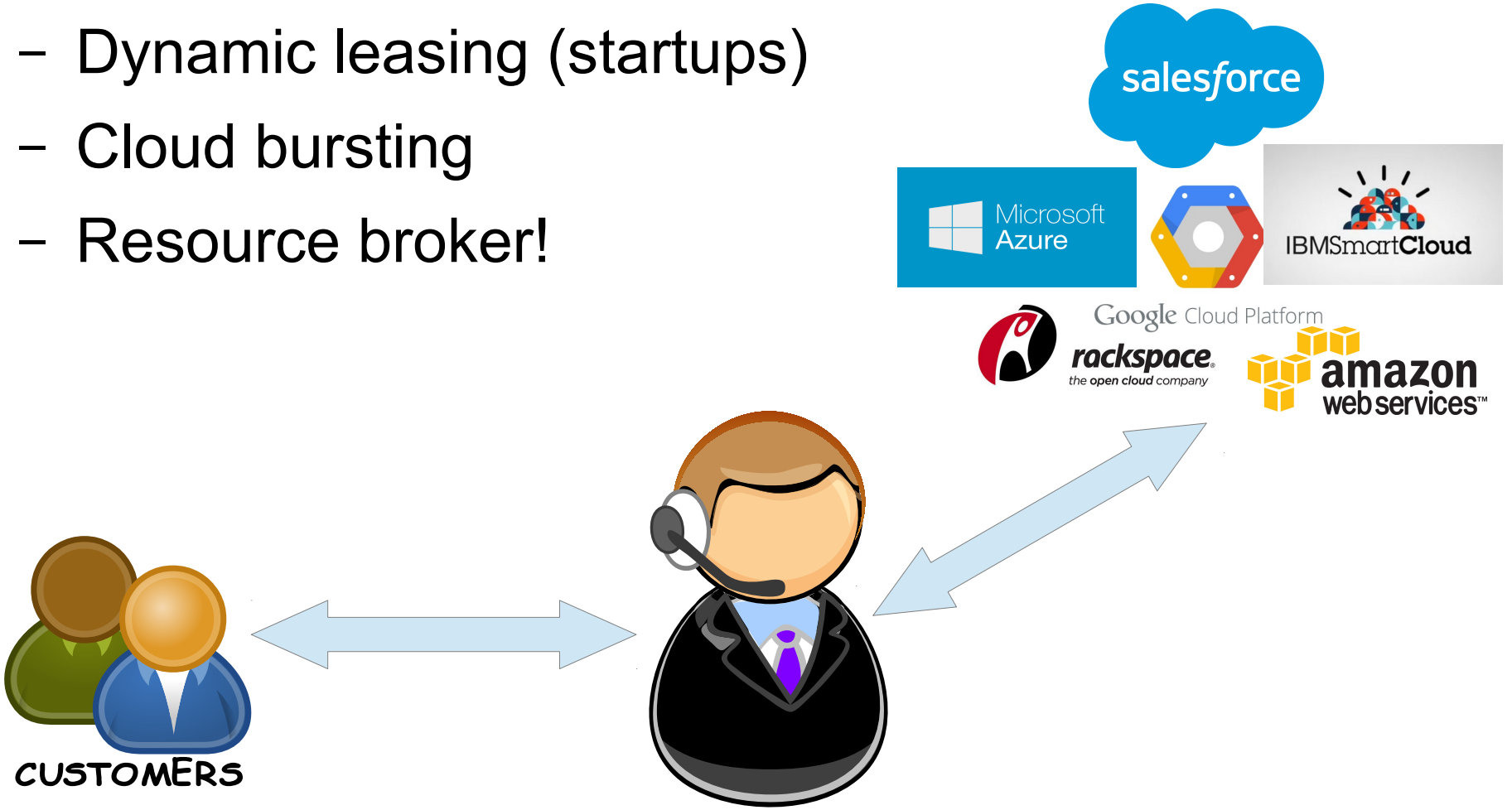
Google Cloud Platform

rackspace
the open cloud company

amazon web services™

Motivation

- Virtualization enables new business models
  - Dynamic leasing (startups)
  - Cloud bursting
  - Resource broker!



Related work

- Ski rental (classical problem):
  - Buy or rent skis?

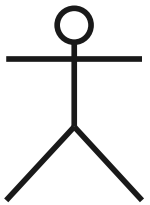
- Parking Permit Problem (Meyerson 2005):
  - Unpredictable car usage.
  - Discount on longer permits.
  - Which duration to buy?

What is a clever way to buy (discounted) resources?

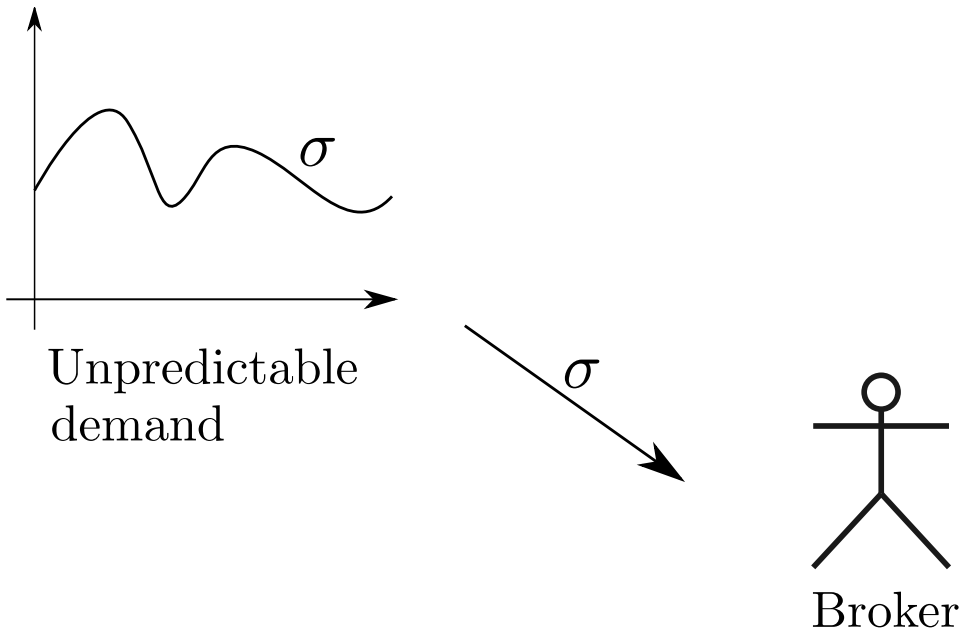
Overview

- Model
- Online Algorithm
- Upper / Lower Bound
- Offline Algorithm
- Conclusion

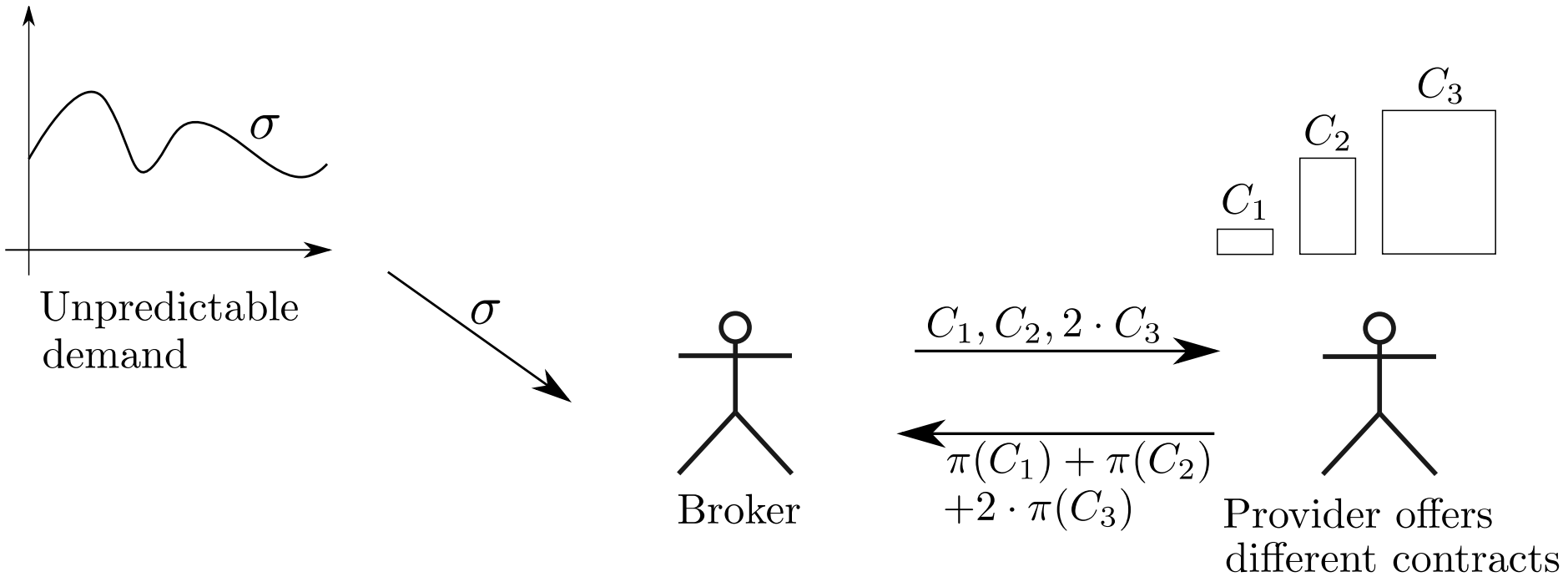
Model



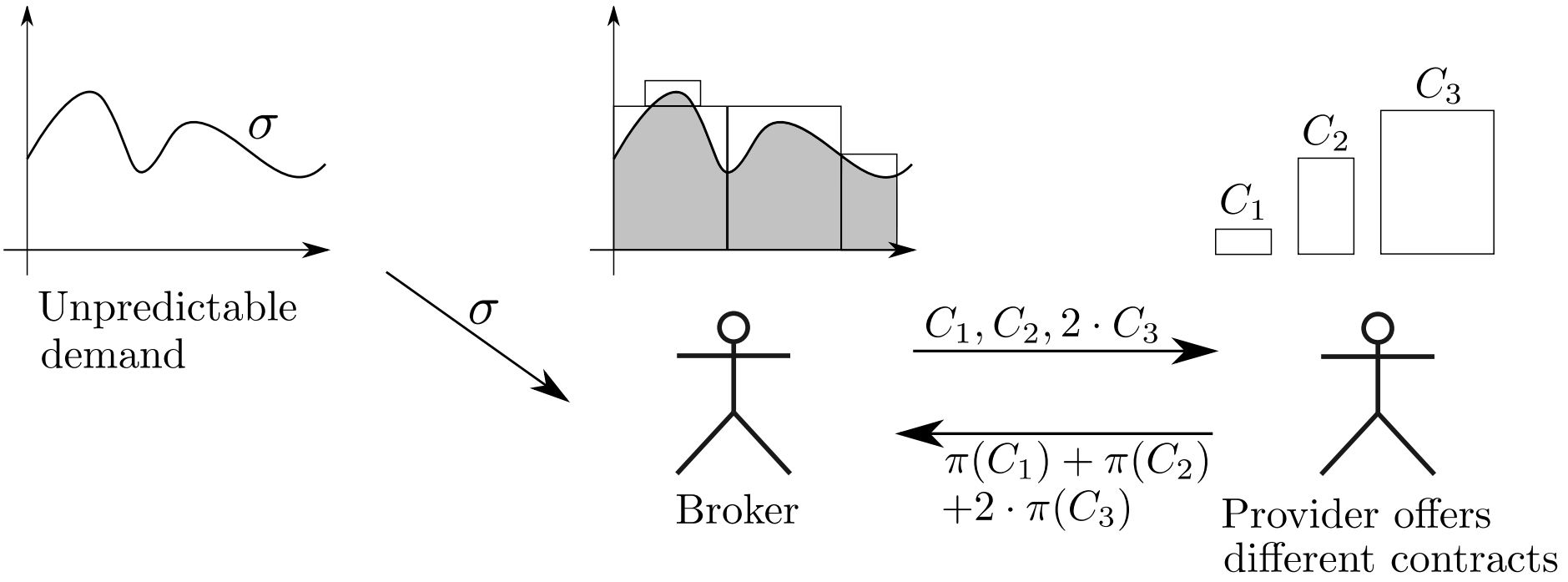
Model



Model

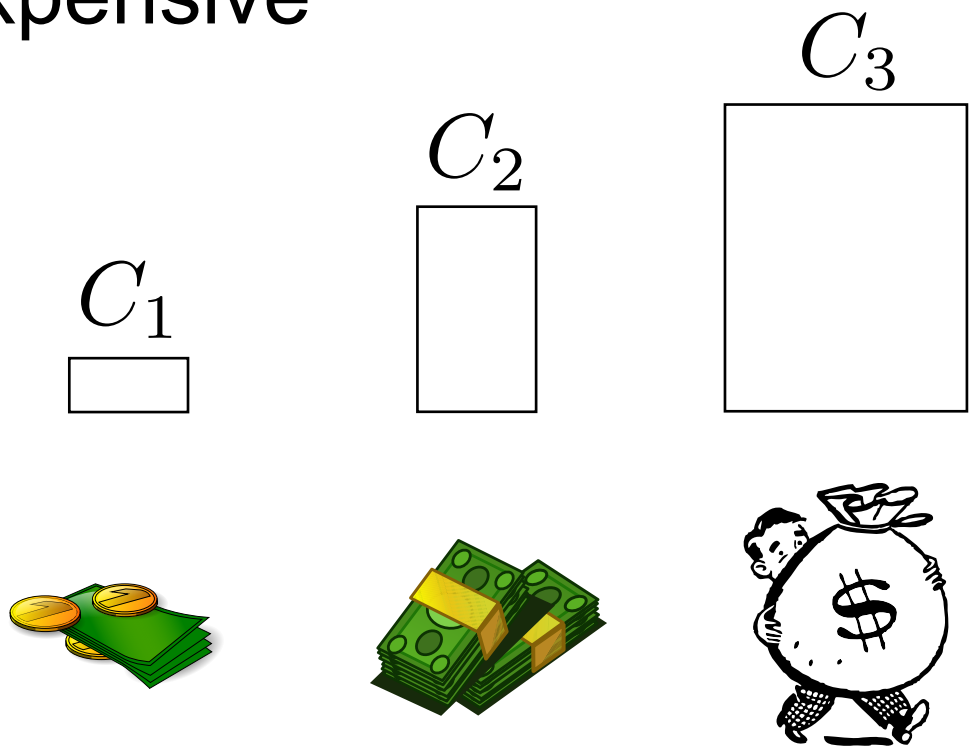

σ
Unpredictable demand
σ
Broker
$C_1, C_2, 2 \cdot C_3$
$\pi(C_1) + \pi(C_2)$
$+2 \cdot \pi(C_3)$
C_1
C_2
C_3
Provider offers different contracts

Model


σ
Unpredictable
demand
σ
C_1
C_2
C_3
$C_1, C_2, 2 \cdot C_3$
$\pi(C_1) + \pi(C_2)$
$+2 \cdot \pi(C_3)$
Broker
Provider offers
different contracts

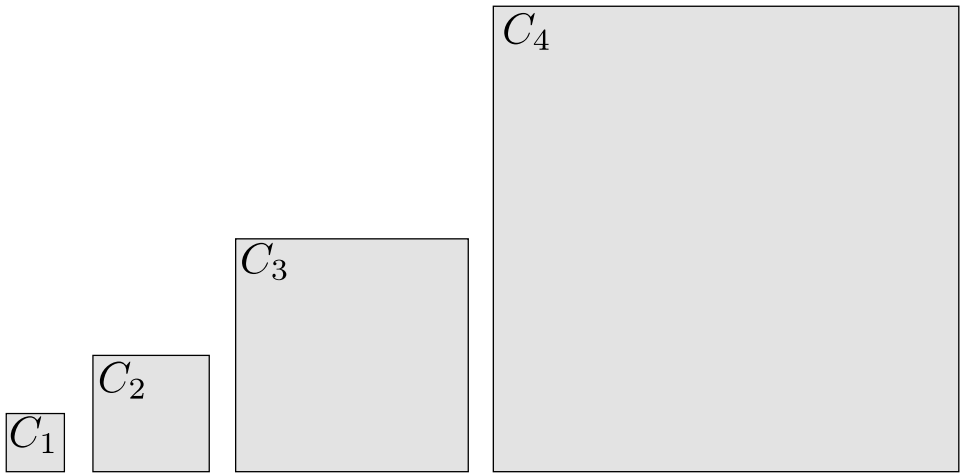
Model - assumptions

1. *Monotonic Prices*, i.e. larger contracts -> more expensive



Model - assumptions

1. *Monotonic Prices*
2. *Multiplicity*, i.e., the next larger contract has a multiple of duration and rate



Model

- Single resource (generalizable to multiple)
- Complete demand needs to be covered
- One lookahead model

- Objective: minimize overall price
  - Minimize competitive ratio

Competitive Strategy for Resource Allocation with Discounts?

Online algorithm – Do's and Don'ts

- Try to maximize the discount

- Do not overestimate the demand

Online algorithm – Do's and Don'ts

- Try to maximize the discount

- Do not overestimate the demand

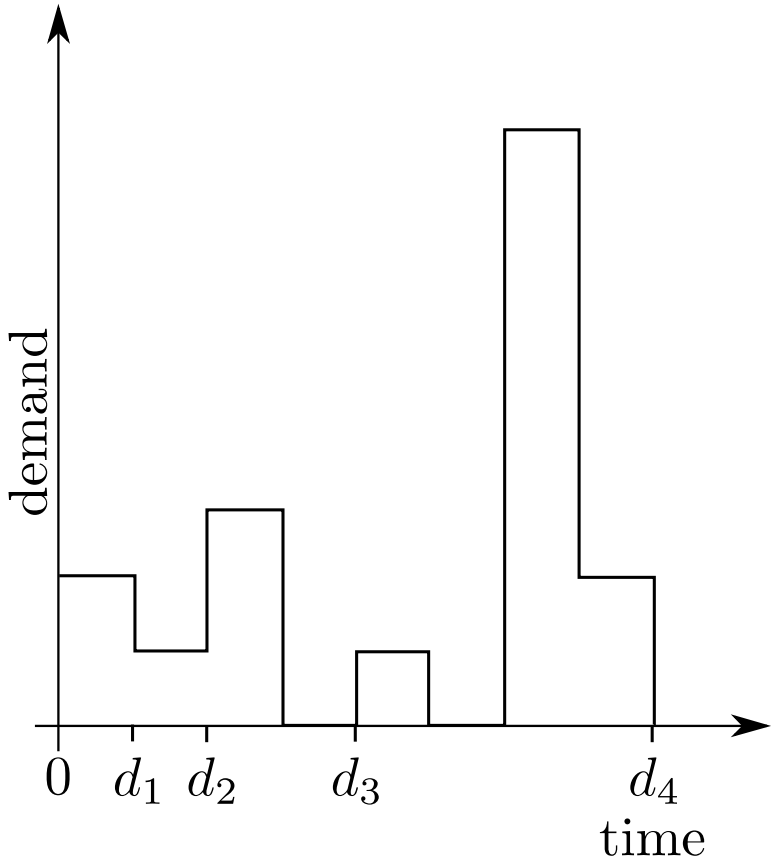
→ Let's stick to someone who should know better.

Online algorithm - ON2D

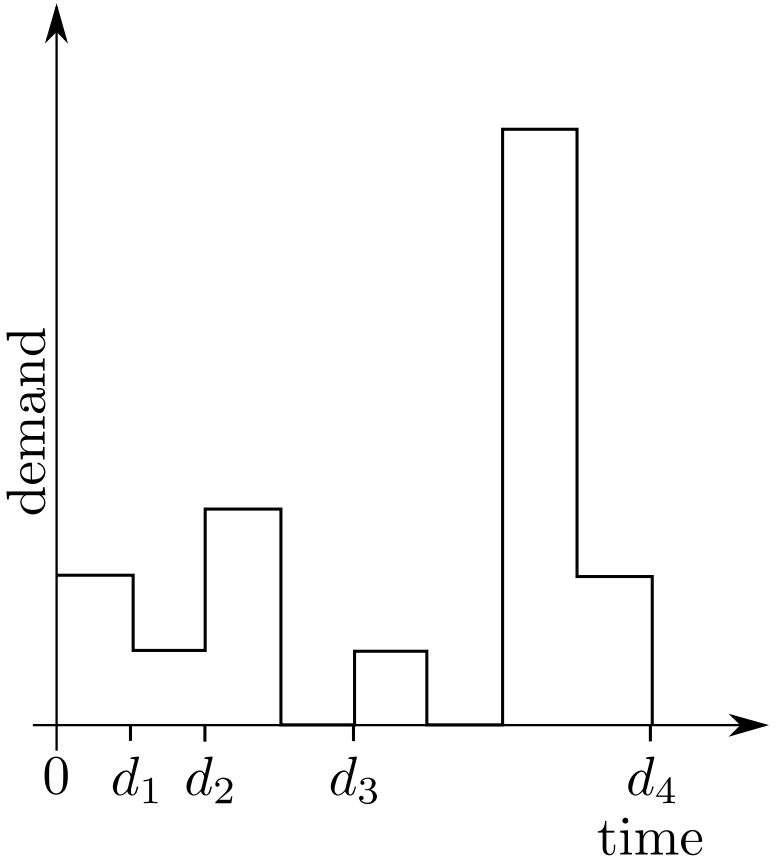
- ON2D mimics the offline algorithm OFF2D:
  1. Check for uncovered demand
  2. Buy the exact same contract OFF2D would buy

Online algorithm - example

Offline

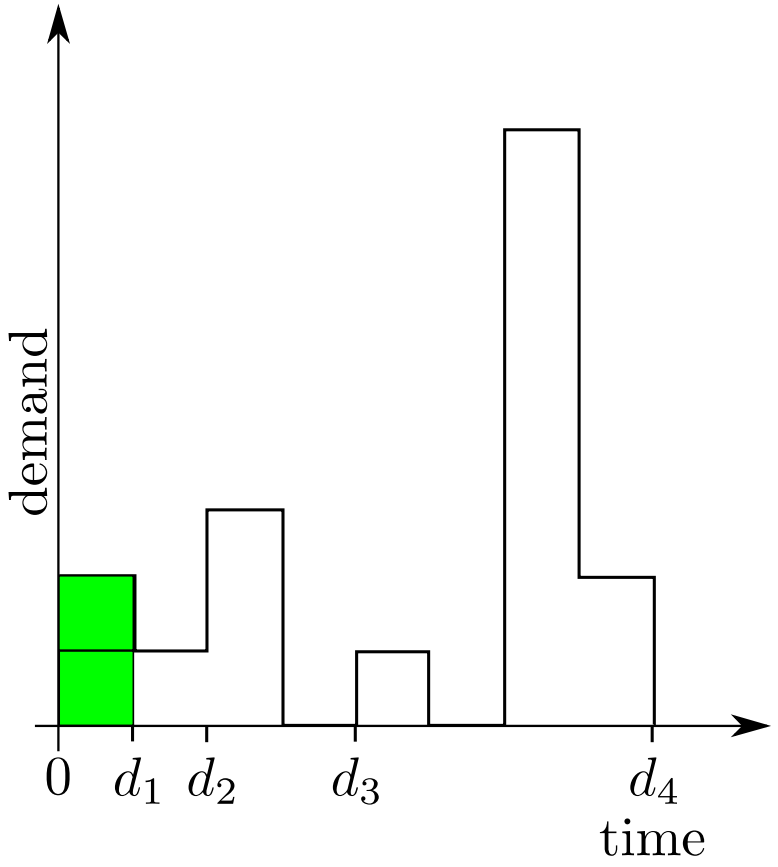


Online

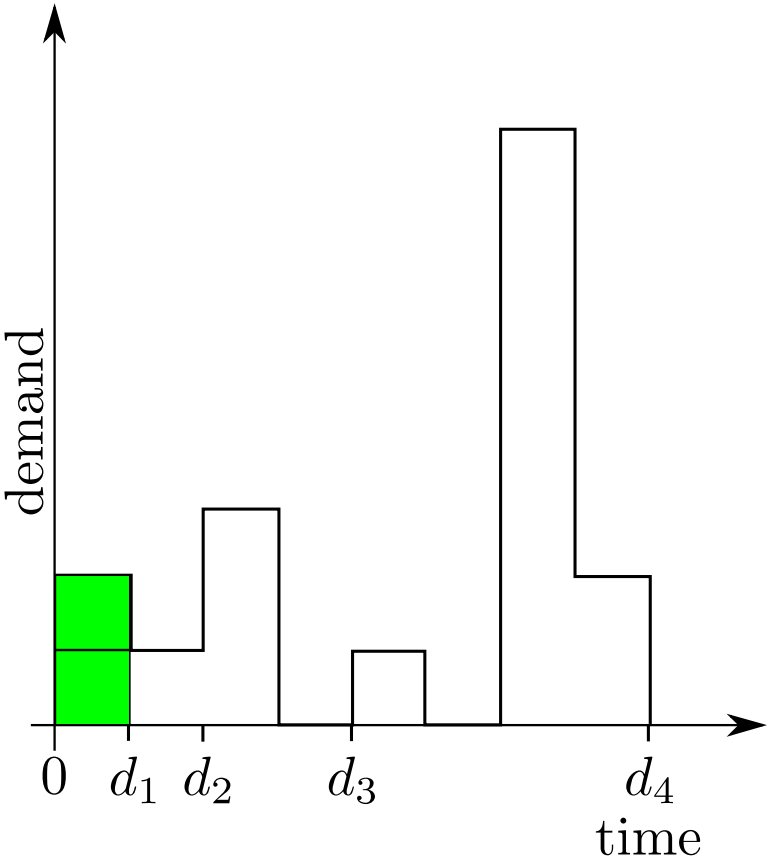


Online algorithm - example

Offline

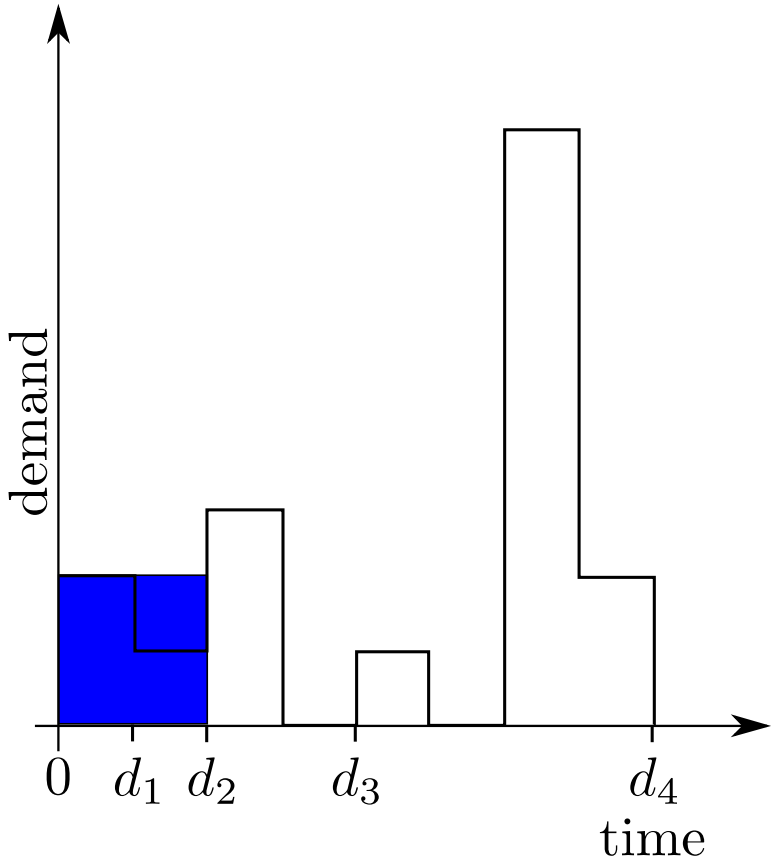


Online

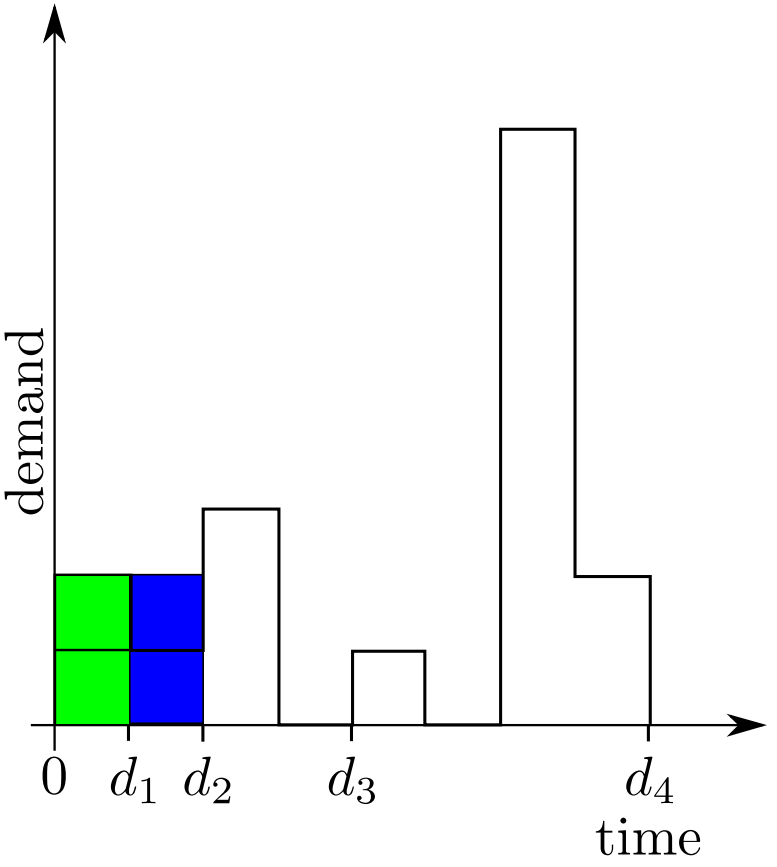


Online algorithm - example

Offline

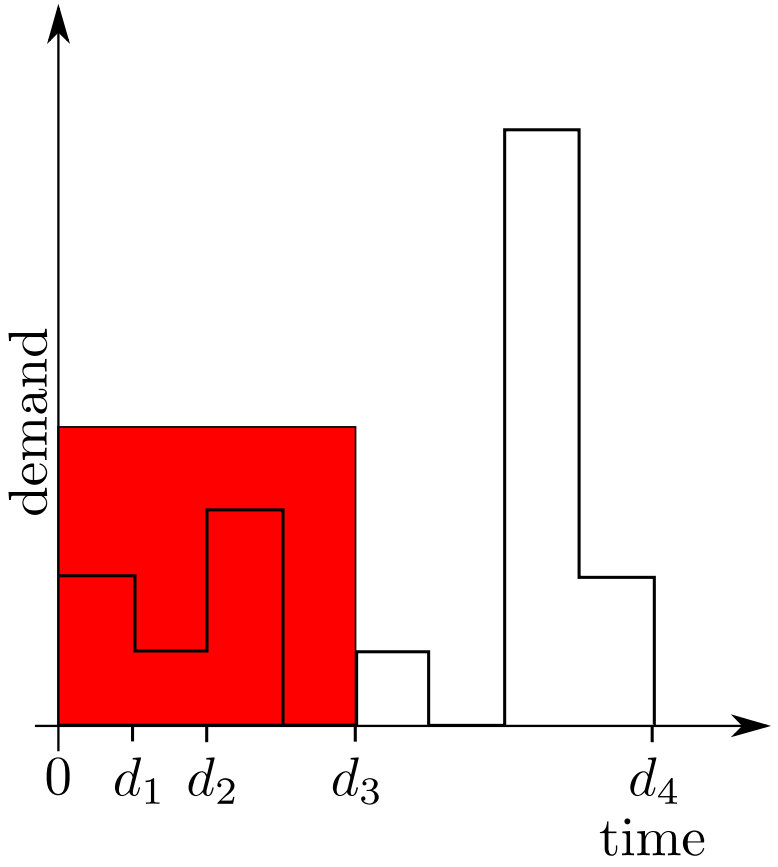


Online

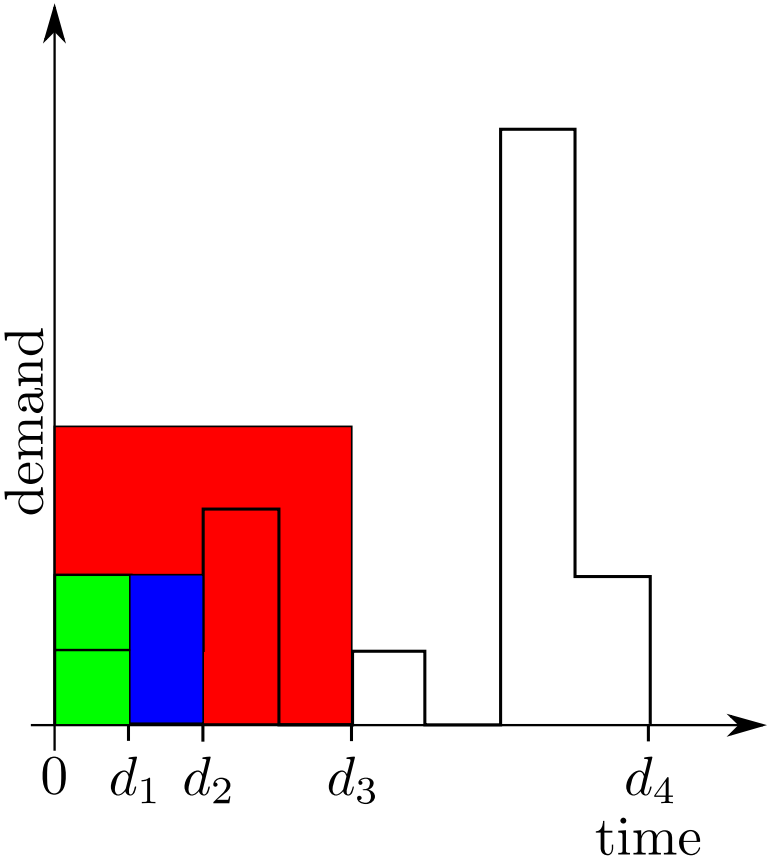


Online algorithm - example

Offline

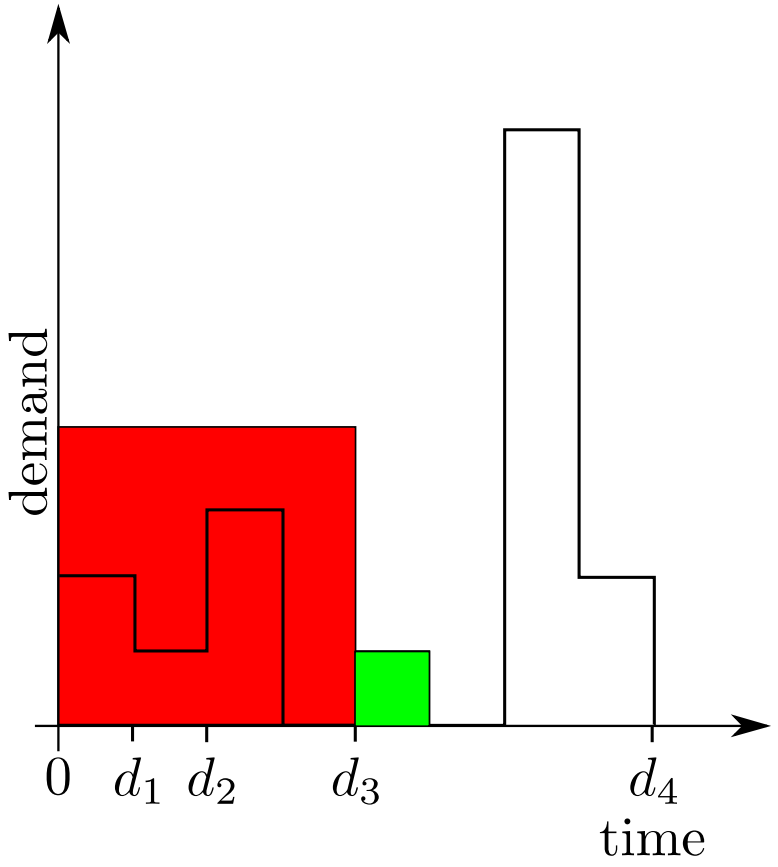


Online

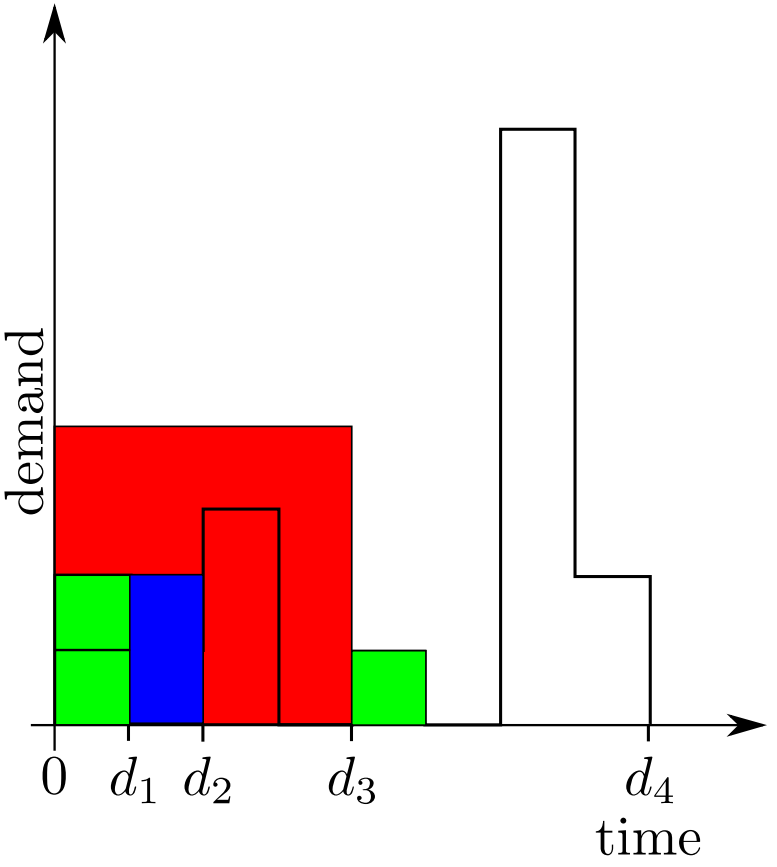


Online algorithm - example

Offline

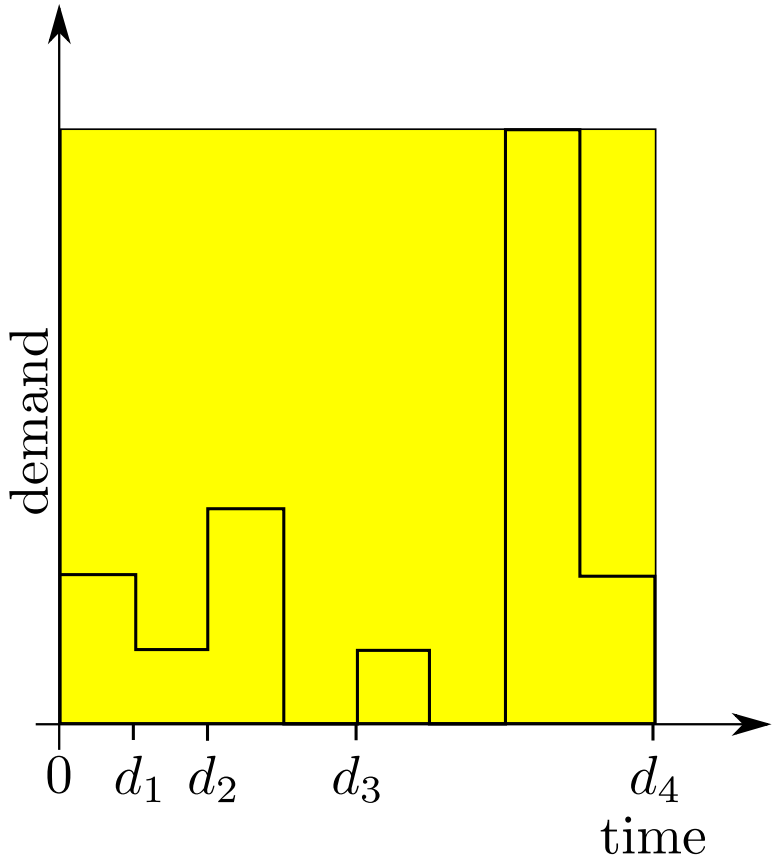


Online

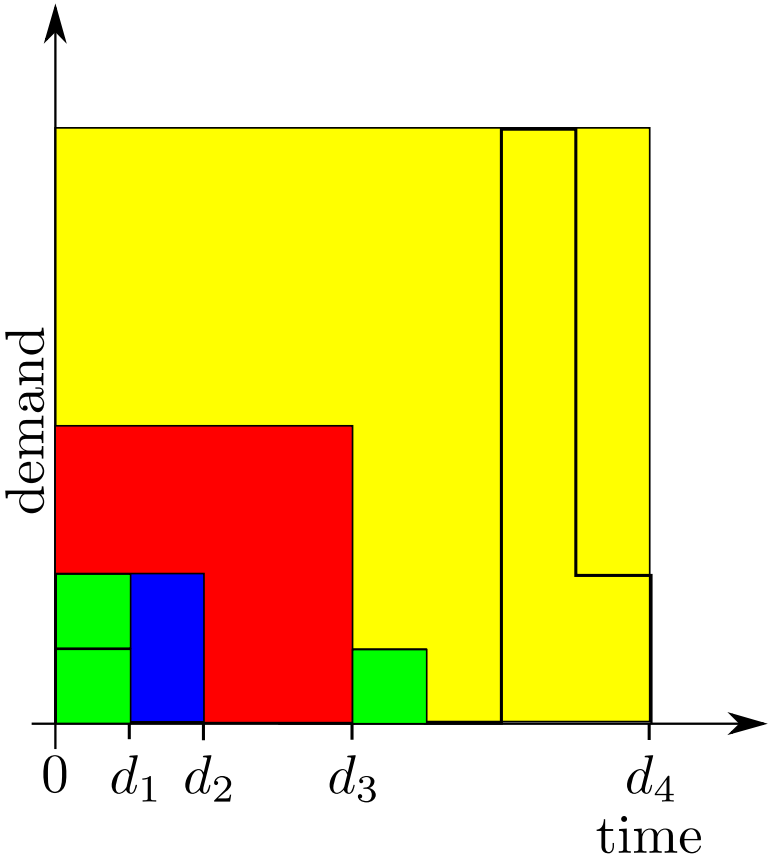


Online algorithm - example

Offline



Online



How good is ON2D?

- Is it competitive?
- Is it close to best?
- Generally applicable?

How good is ON2D?

- Is it competitive?

k-competitive!
(*k* = numbers of contracts)

- Is it close to best?

- Generally applicable?

How good is ON2D?

- Is it competitive?

k-competitive!
(*k* = numbers of contracts)

- Is it close to best?

Lower bound: *k*/3

- Generally applicable?

How good is ON2D?

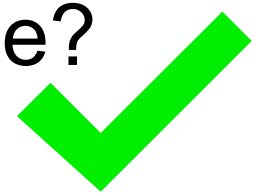
- Is it competitive?

k-competitive!
(*k* = numbers of contracts)

- Is it close to best?

Lower bound: *k*/3

- Generally applicable?



Independent of the discount function!

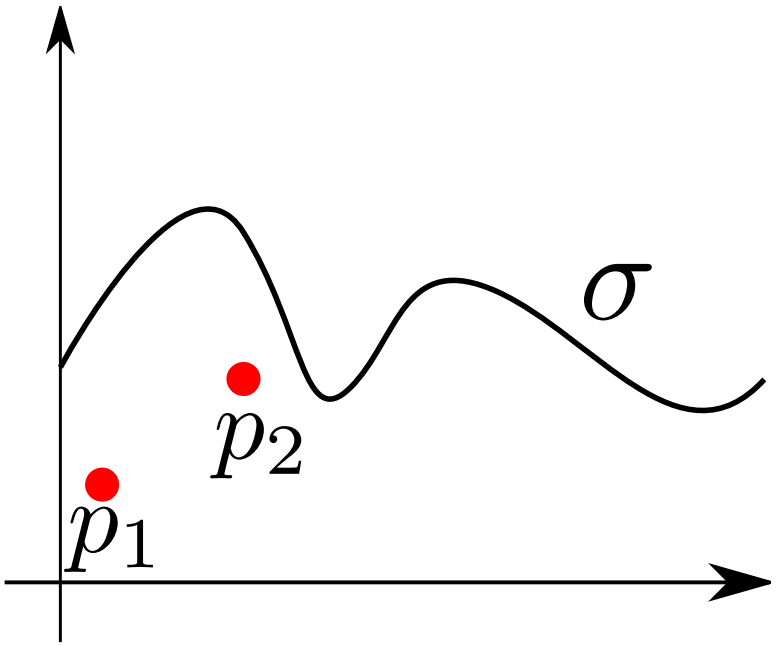
Upper bound

1. Contract independence
2. Worst case classification
3. Maximum cumulative price

$\rightarrow$ *k*-competitive (*k* = number of contracts)

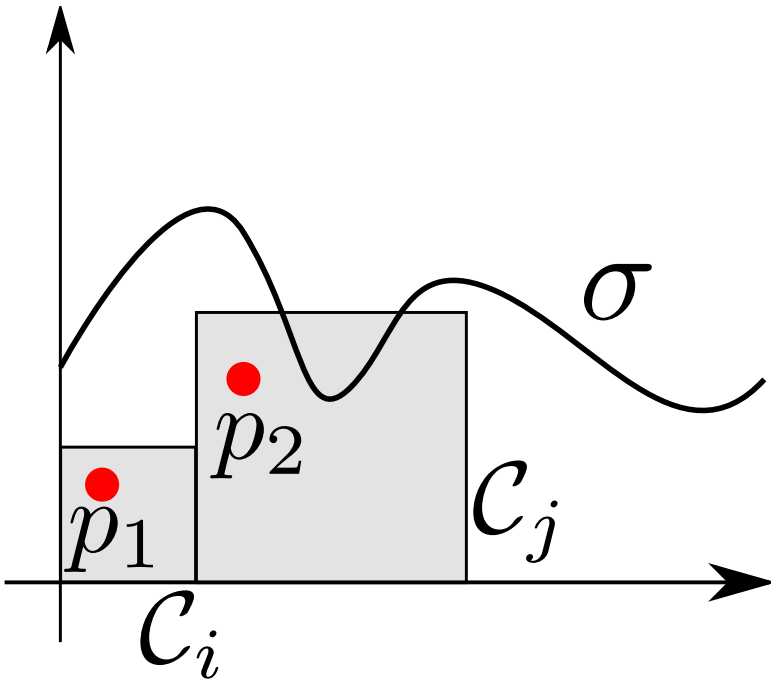
Upper bound contract independence

- p_1
- p_2



Upper bound
contract independence

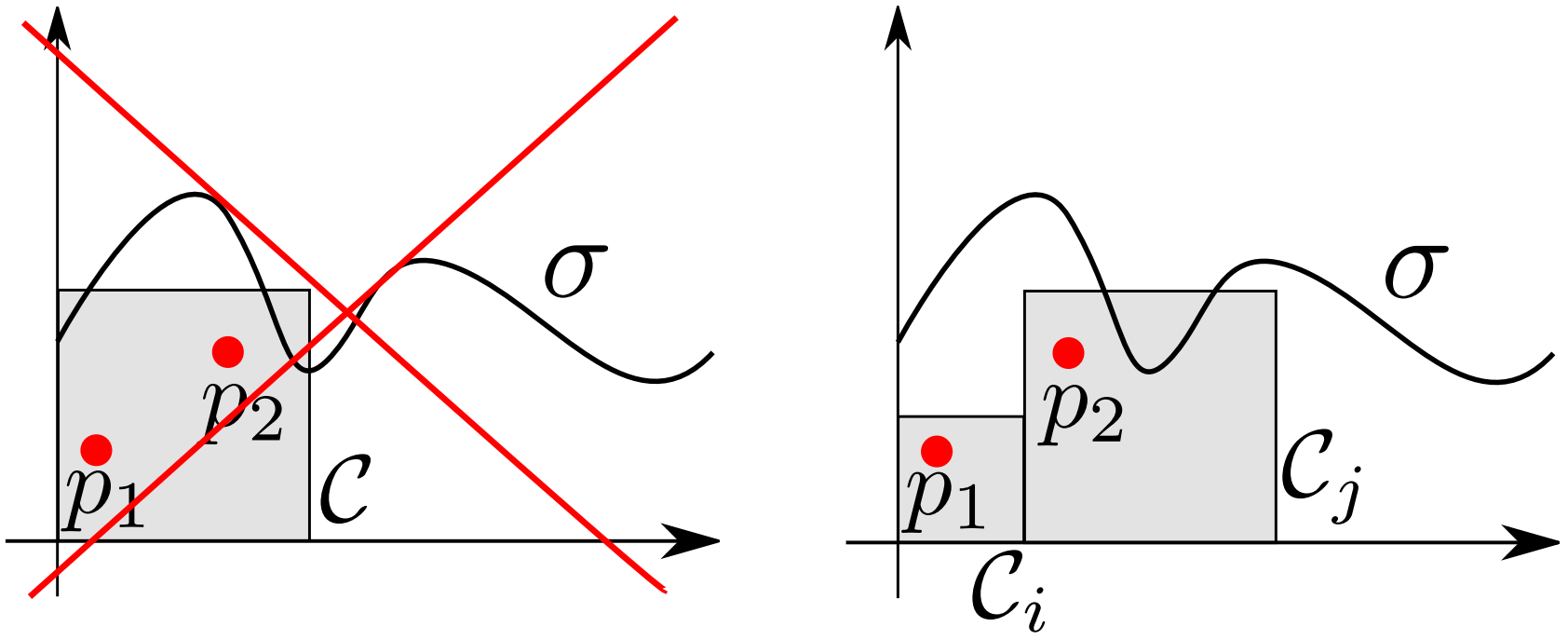
- p_1 covered by $\mathcal{C}_i$, at time t
- p_2 covered by $\mathcal{C}_j$, at time t



Upper bound contract independence

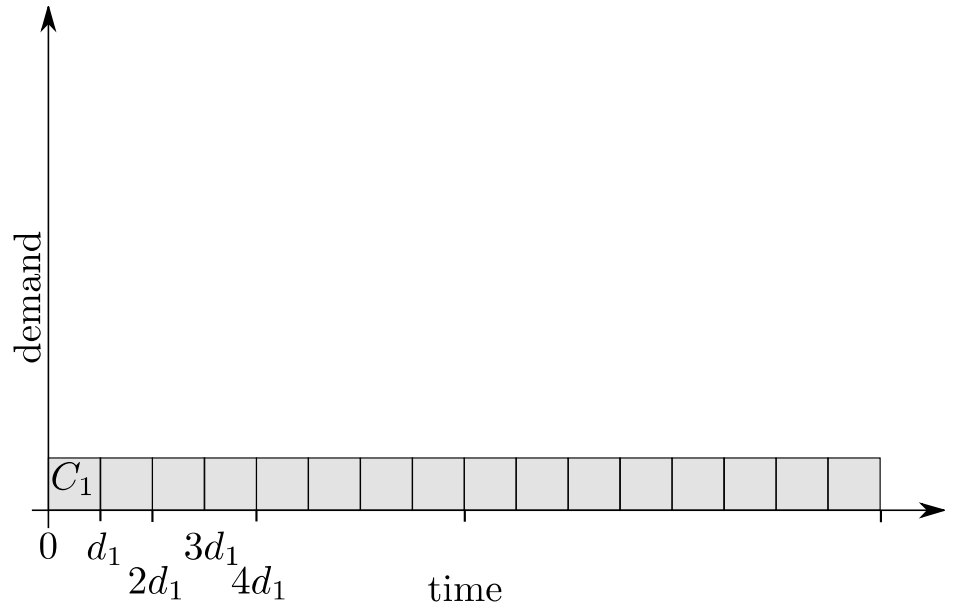
- p_1 covered by $\mathcal{C}_i$, at time t
- p_2 covered by $\mathcal{C}_j$, at time t

  $\rightarrow$ No contract $\mathcal{C}$ at time $t' < t$ such that, $\mathcal{C}$ covers both p_1 and p_2



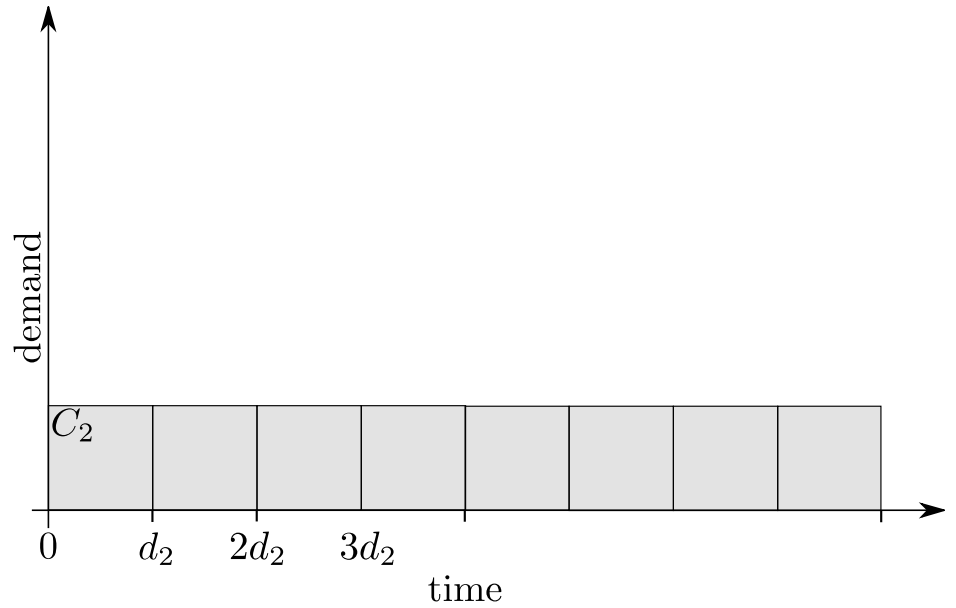
Upper bound interval assumption

- Contracts of length *d* start only every *d* time



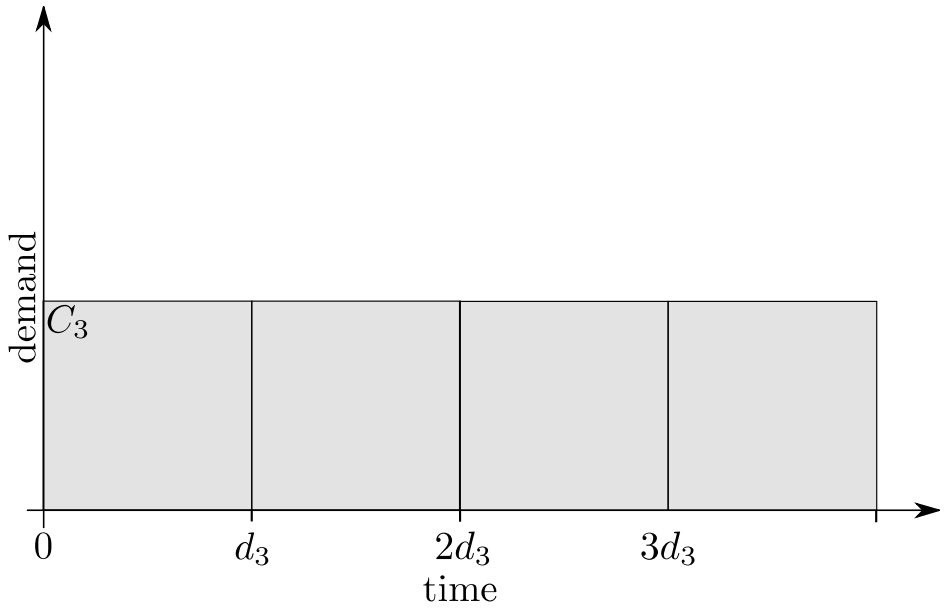
Upper bound interval assumption

- Contracts of length *d* start only every *d* time



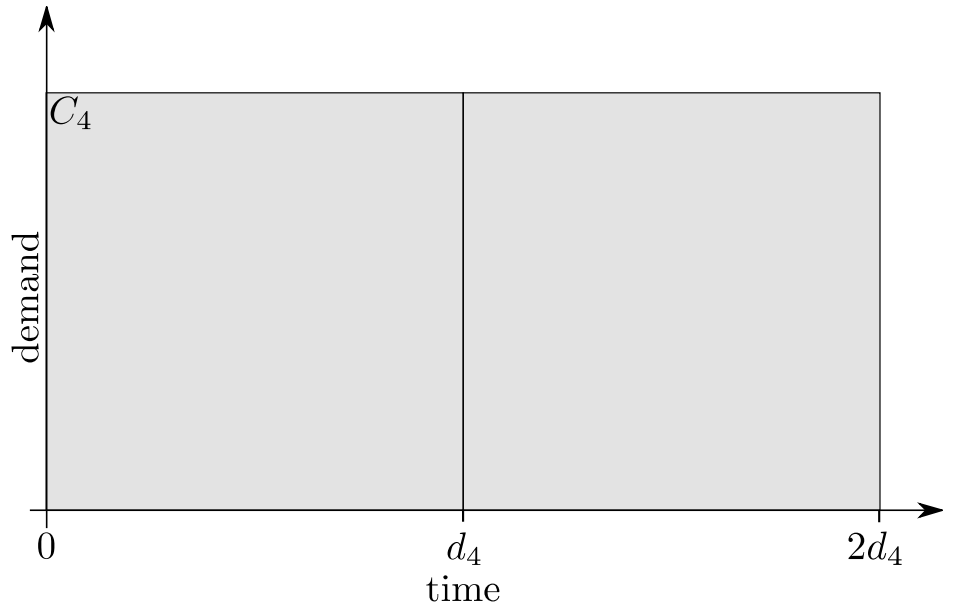
Upper bound interval assumption

- Contracts of length *d* start only every *d* time



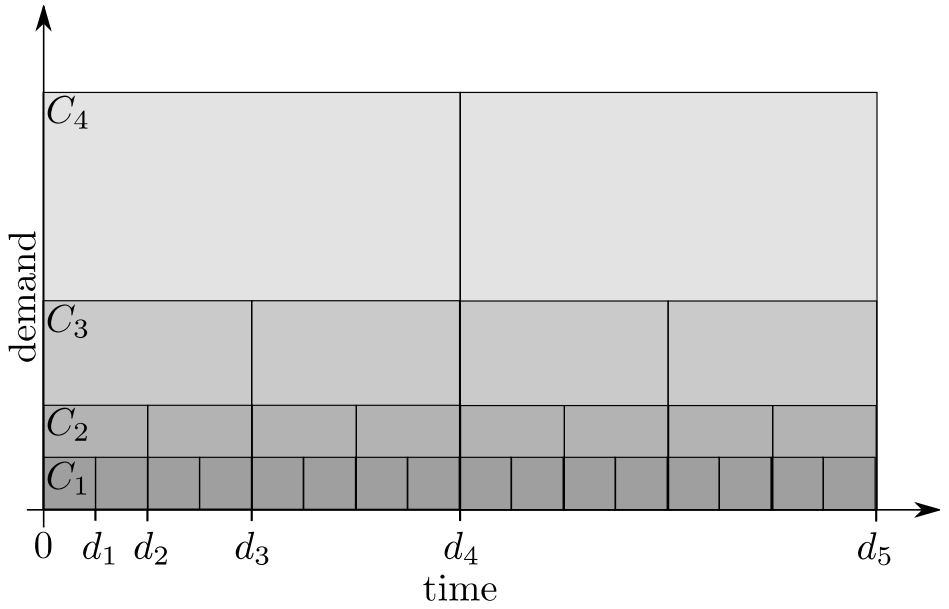
Upper bound interval assumption

- Contracts of length *d* start only every *d* time



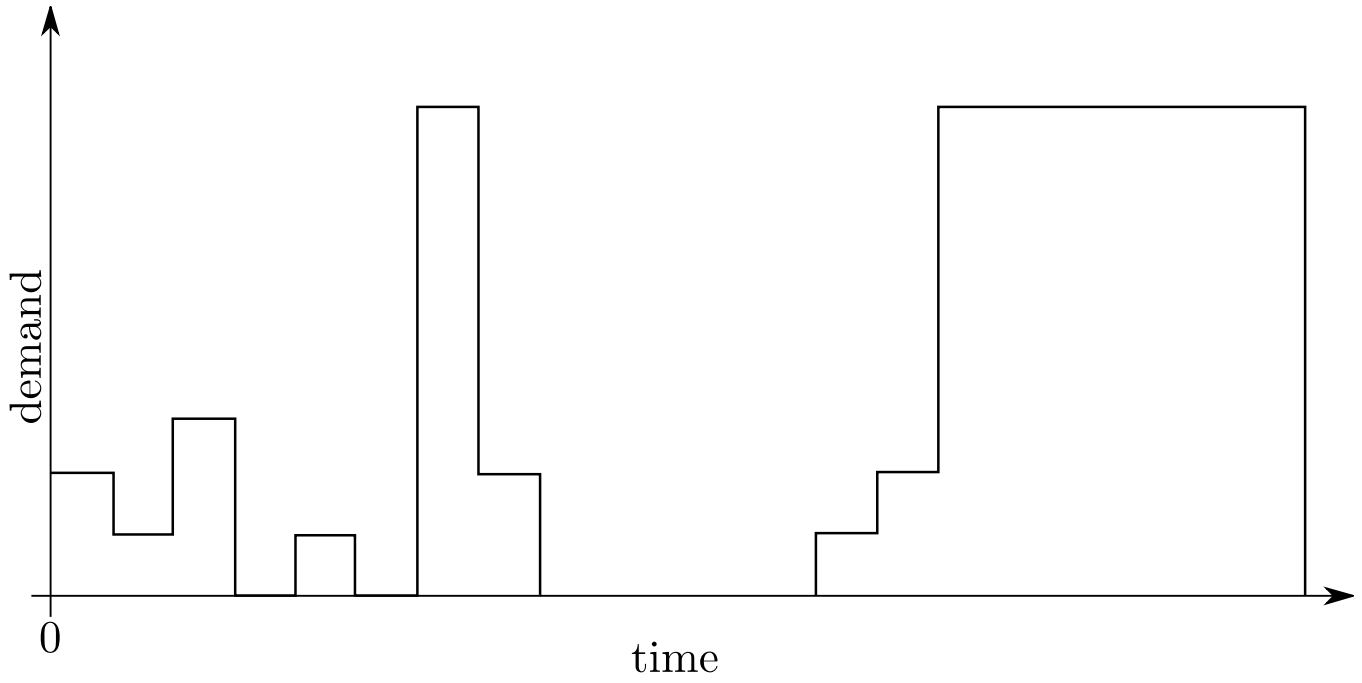
Upper bound interval assumption

- Contracts of length *d* start only every *d* time



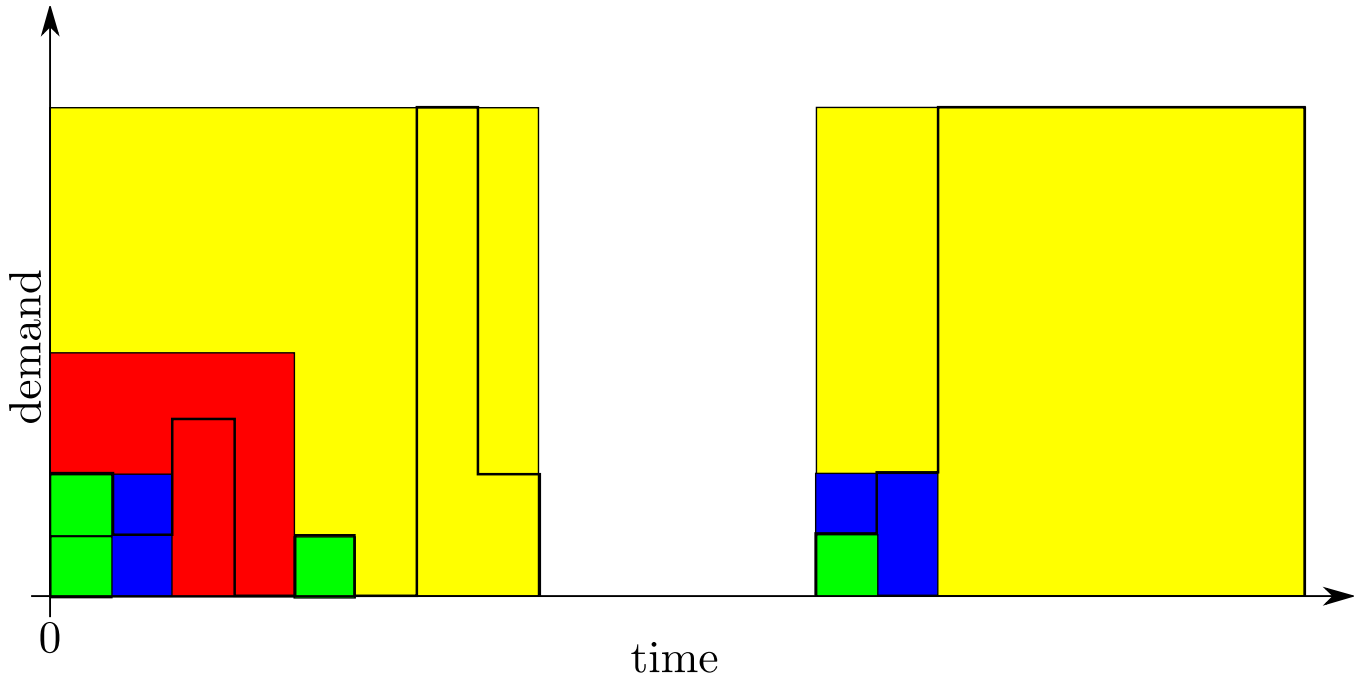
Upper bound
worst case

- Worst case: OFF2D buys only one contract



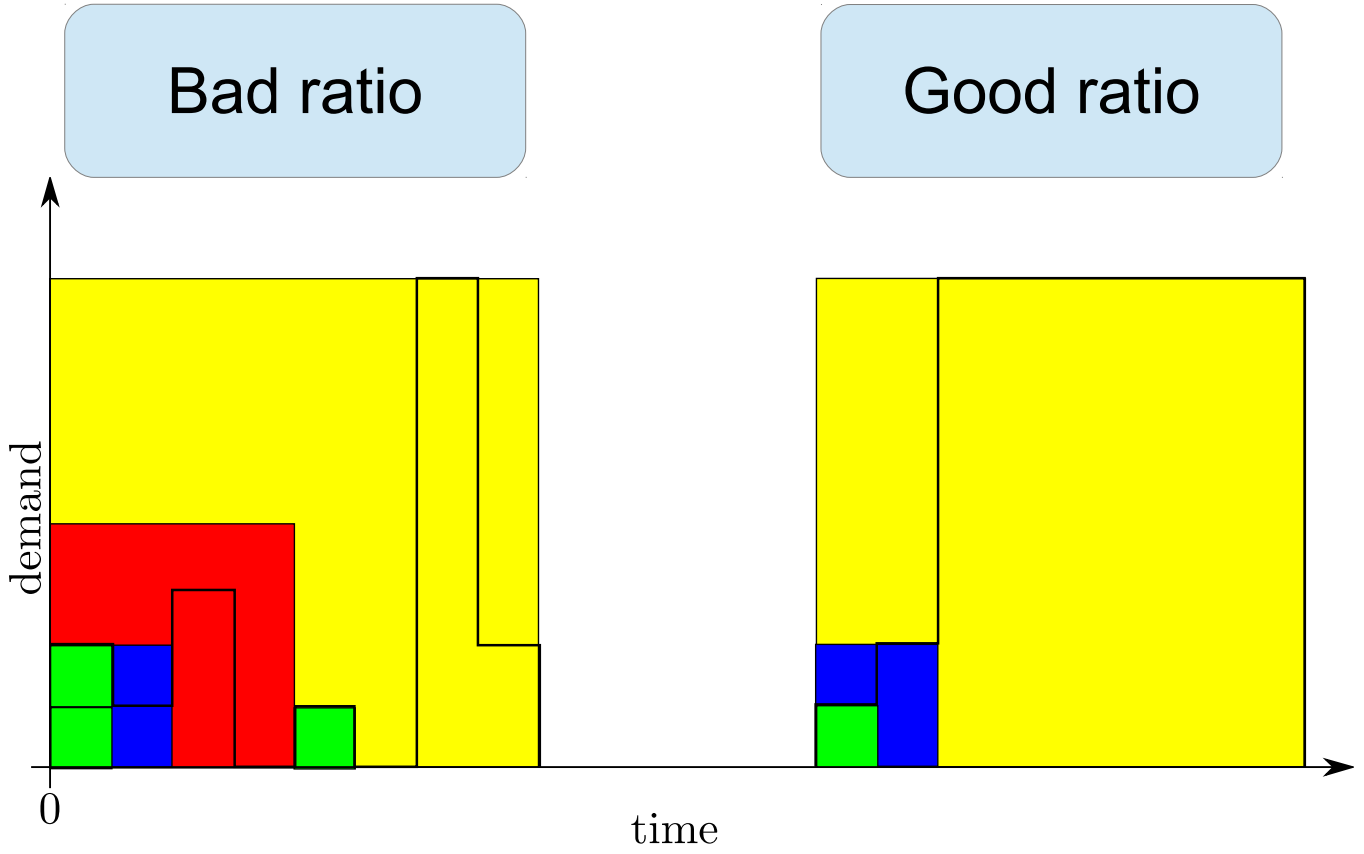
Upper bound
worst case

- Worst case: OFF2D buys only one contract



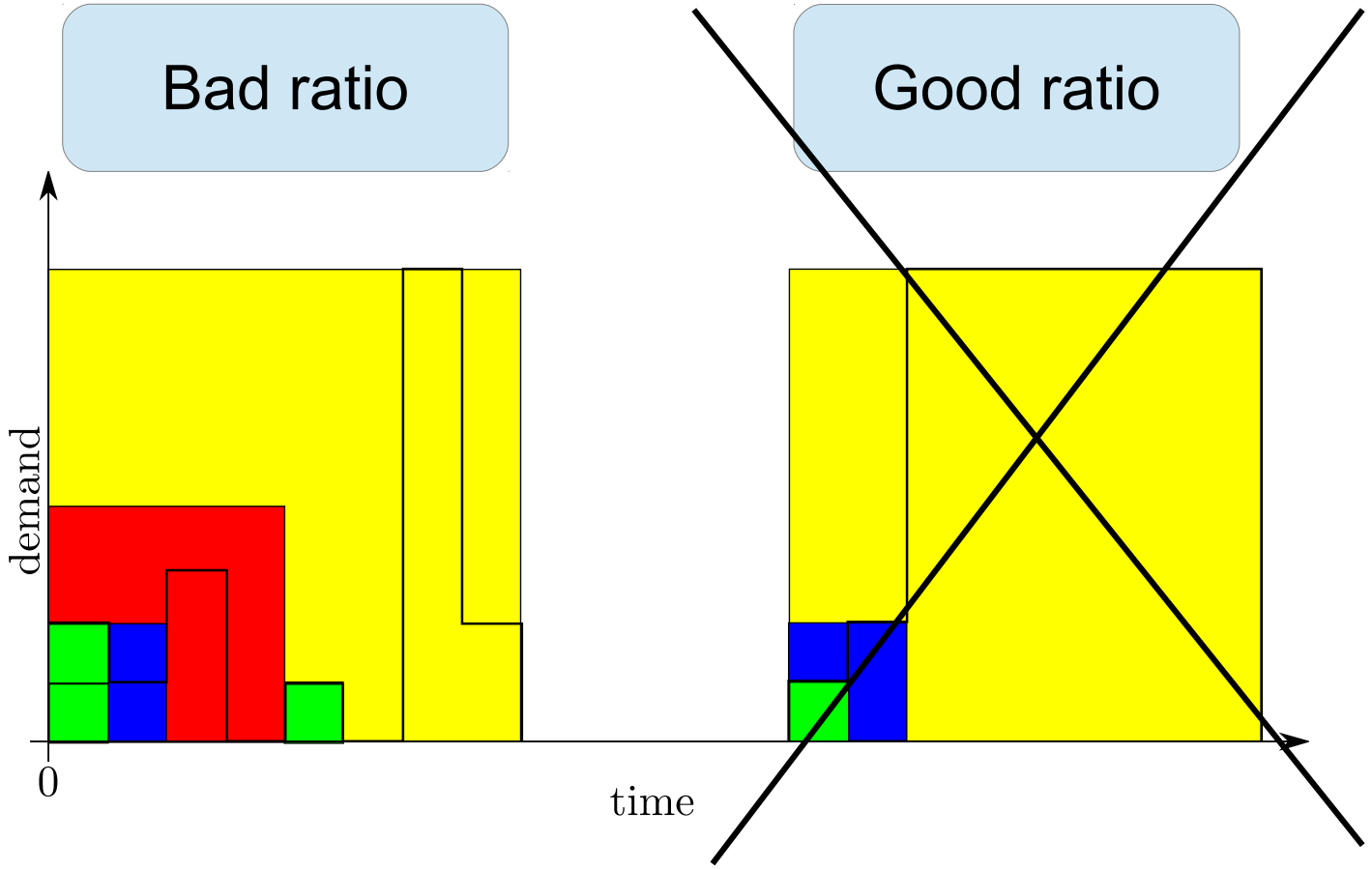
Upper bound
worst case

- Worst case: OFF2D buys only one contract



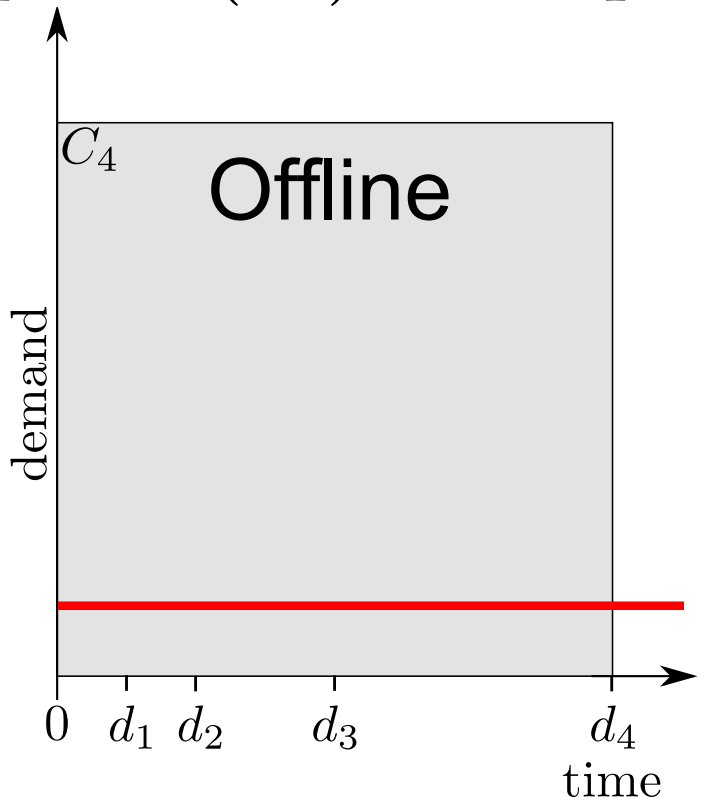
Upper bound worst case

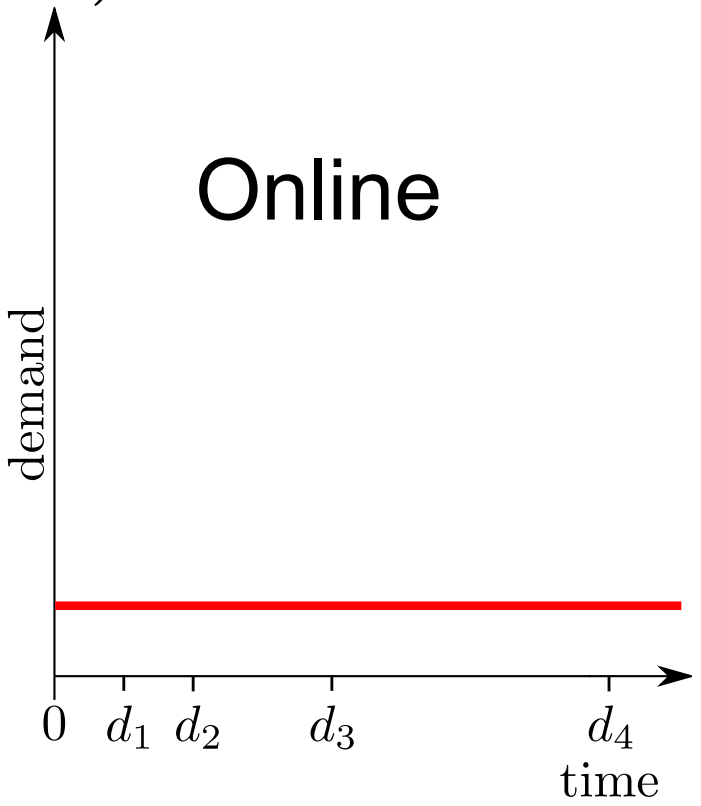
- Worst case: OFF2D buys only one contract



Upper bound
worst case - example

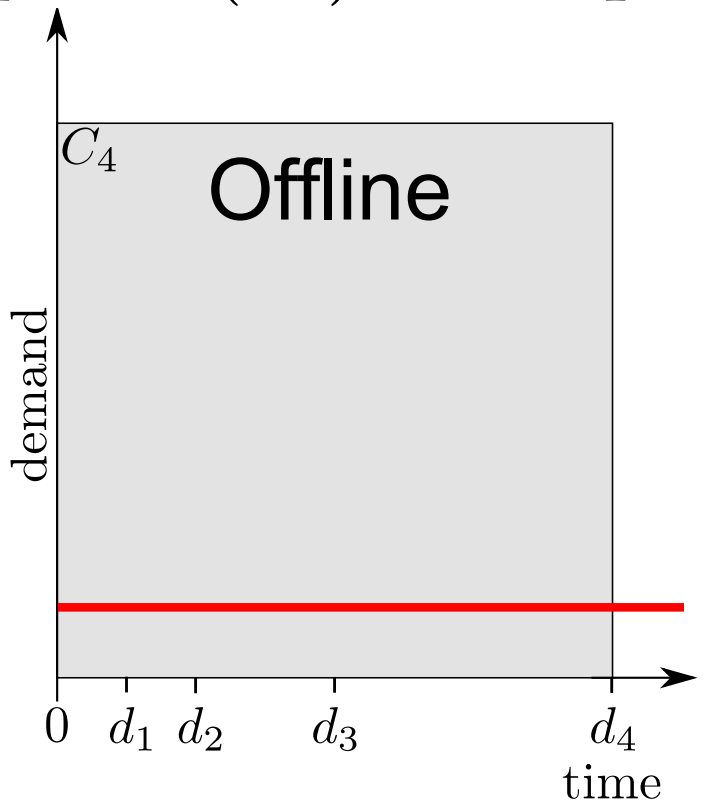
- Uniform demand of 1 per timeslot
- $\mathcal{C}_i = (2^{i-1}, 2^{i-1})$
- $price(\mathcal{C}_i) = 2 \cdot price(\mathcal{C}_{i-1})$

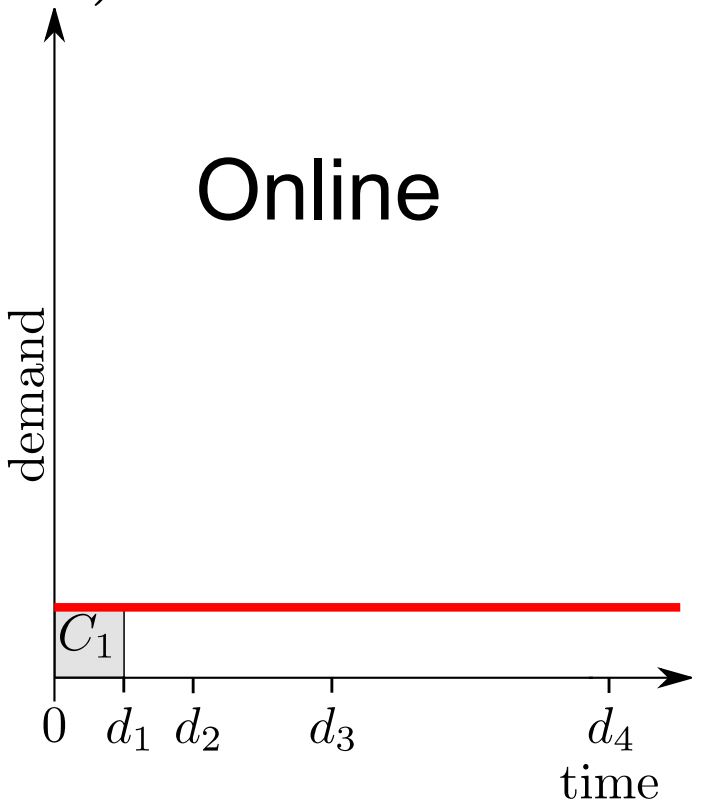




Upper bound
worst case - example

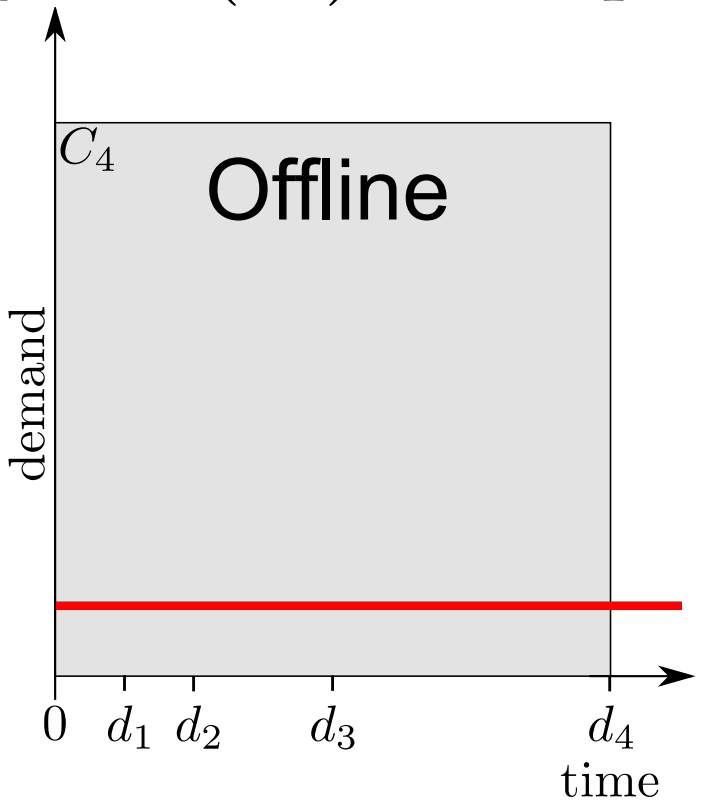
- Uniform demand of 1 per timeslot
- $\mathcal{C}_i = (2^{i-1}, 2^{i-1})$
- $price(\mathcal{C}_i) = 2 \cdot price(\mathcal{C}_{i-1})$

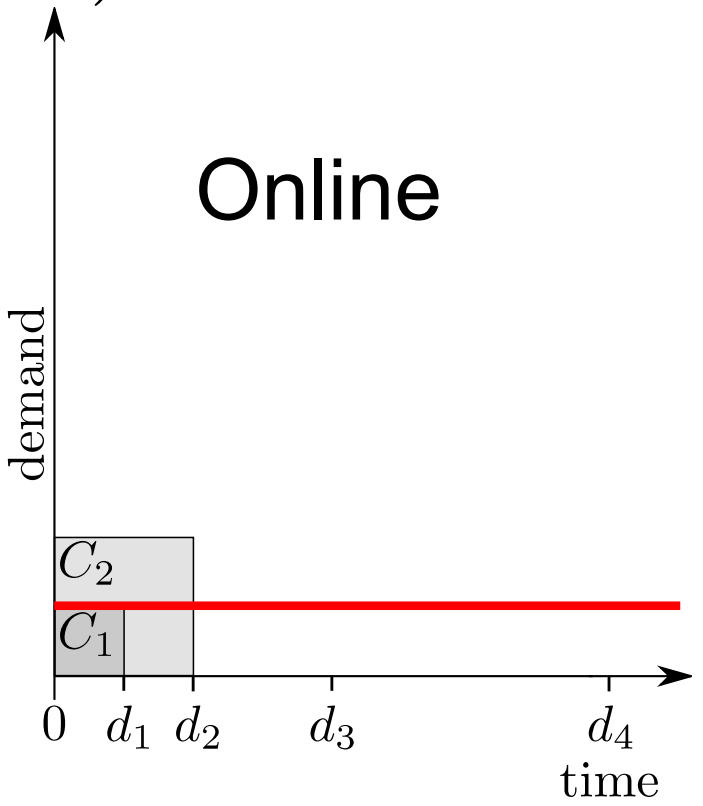




Upper bound
worst case - example

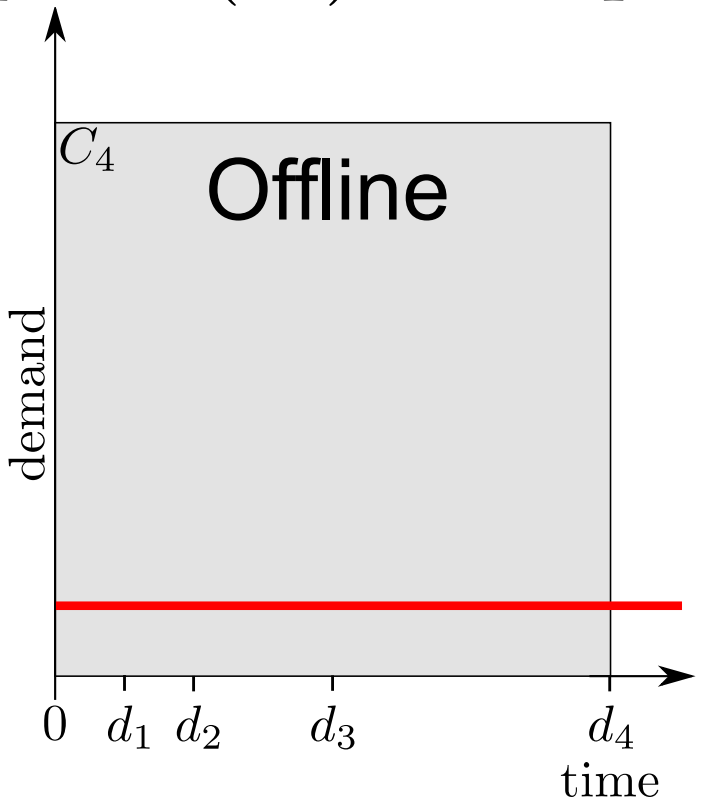
- Uniform demand of 1 per timeslot
- $\mathcal{C}_i = (2^{i-1}, 2^{i-1})$
- $price(\mathcal{C}_i) = 2 \cdot price(\mathcal{C}_{i-1})$

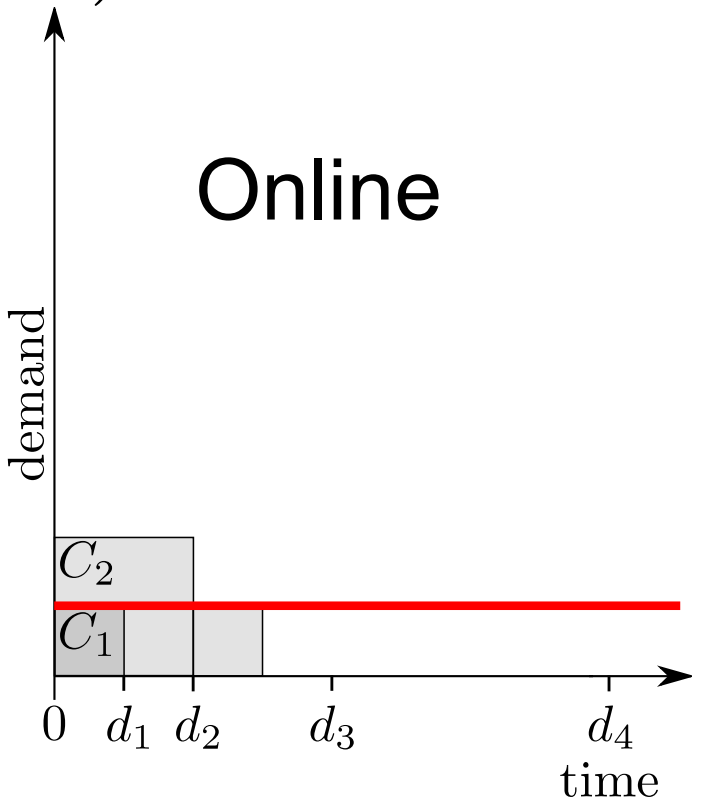




Upper bound
worst case - example

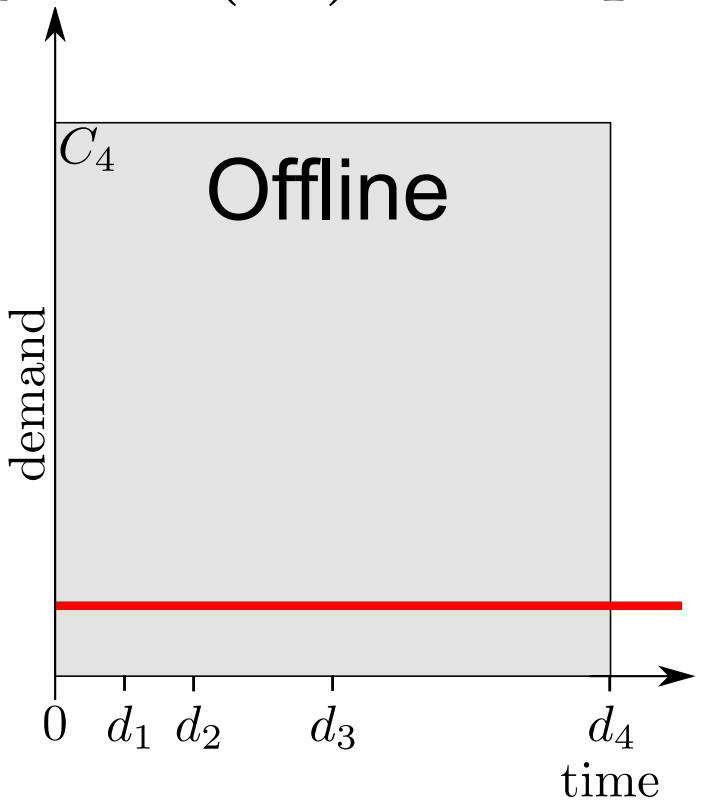
- Uniform demand of 1 per timeslot
- $\mathcal{C}_i = (2^{i-1}, 2^{i-1})$
- $price(\mathcal{C}_i) = 2 \cdot price(\mathcal{C}_{i-1})$

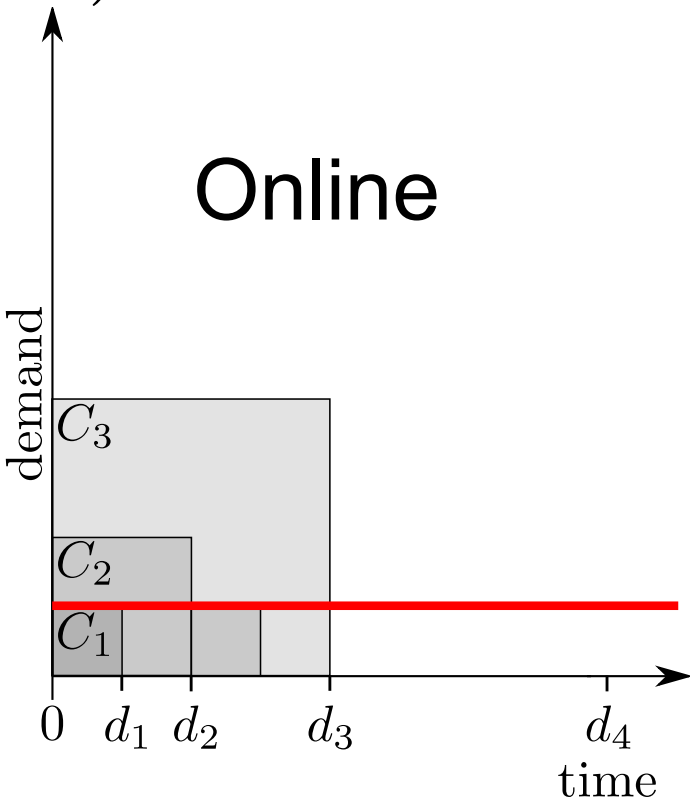




Upper bound
worst case - example

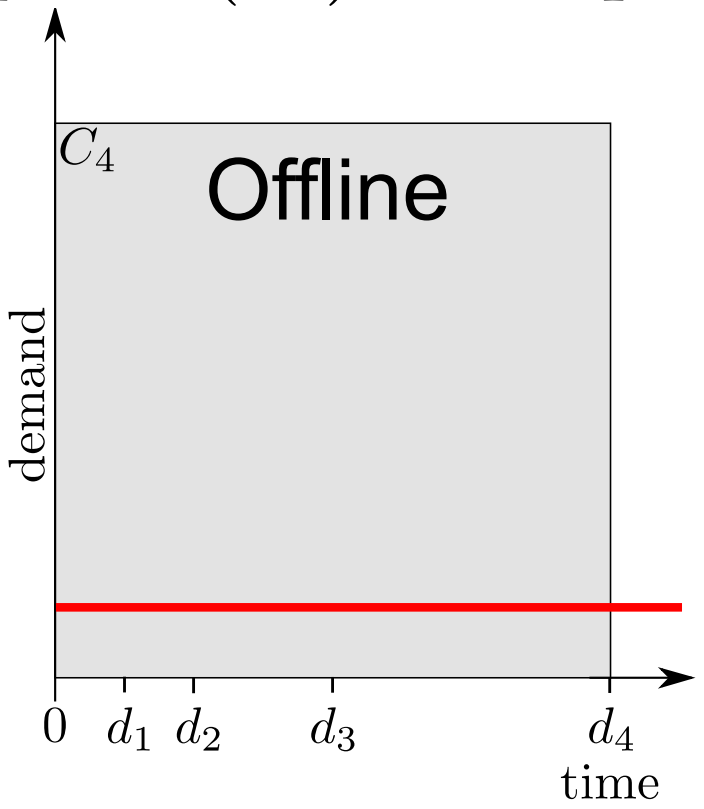
- Uniform demand of 1 per timeslot
- $\mathcal{C}_i = (2^{i-1}, 2^{i-1})$
- $price(\mathcal{C}_i) = 2 \cdot price(\mathcal{C}_{i-1})$

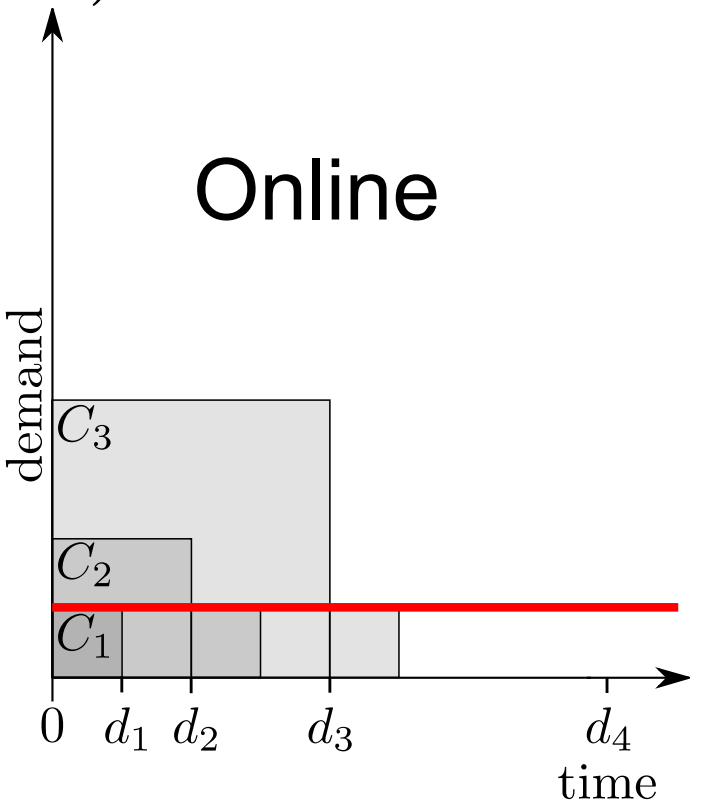




Upper bound
worst case - example

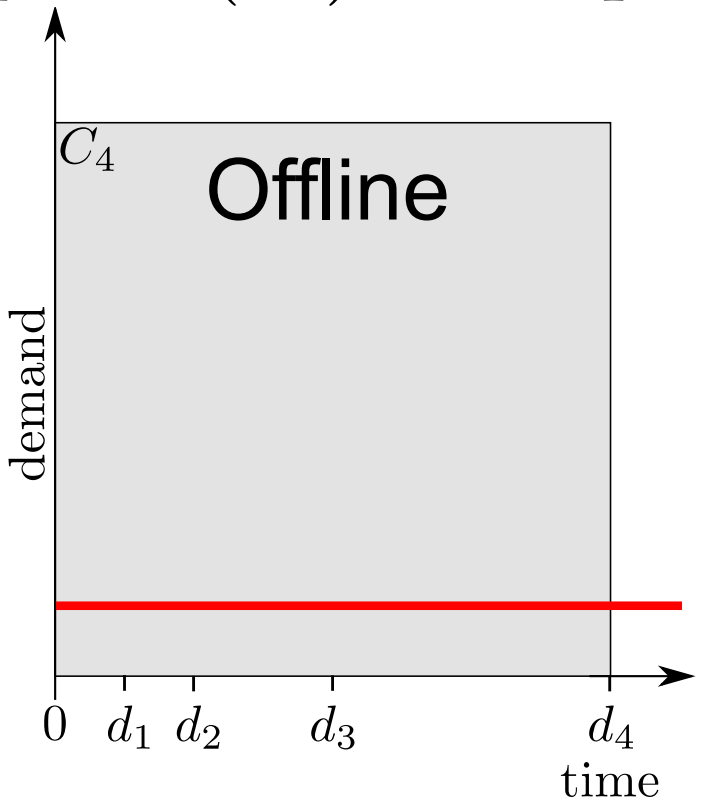
- Uniform demand of 1 per timeslot
- $\mathcal{C}_i = (2^{i-1}, 2^{i-1})$
- $price(\mathcal{C}_i) = 2 \cdot price(\mathcal{C}_{i-1})$

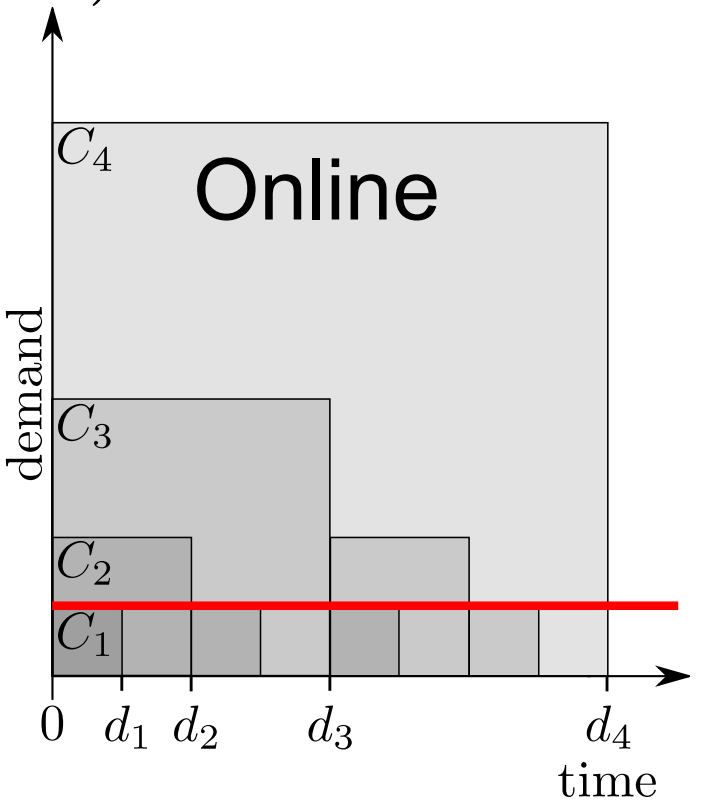




Upper bound
worst case - example

- Uniform demand of 1 per timeslot
- $\mathcal{C}_i = (2^{i-1}, 2^{i-1})$
- $price(\mathcal{C}_i) = 2 \cdot price(\mathcal{C}_{i-1})$





Upper bound maximum cumulative price

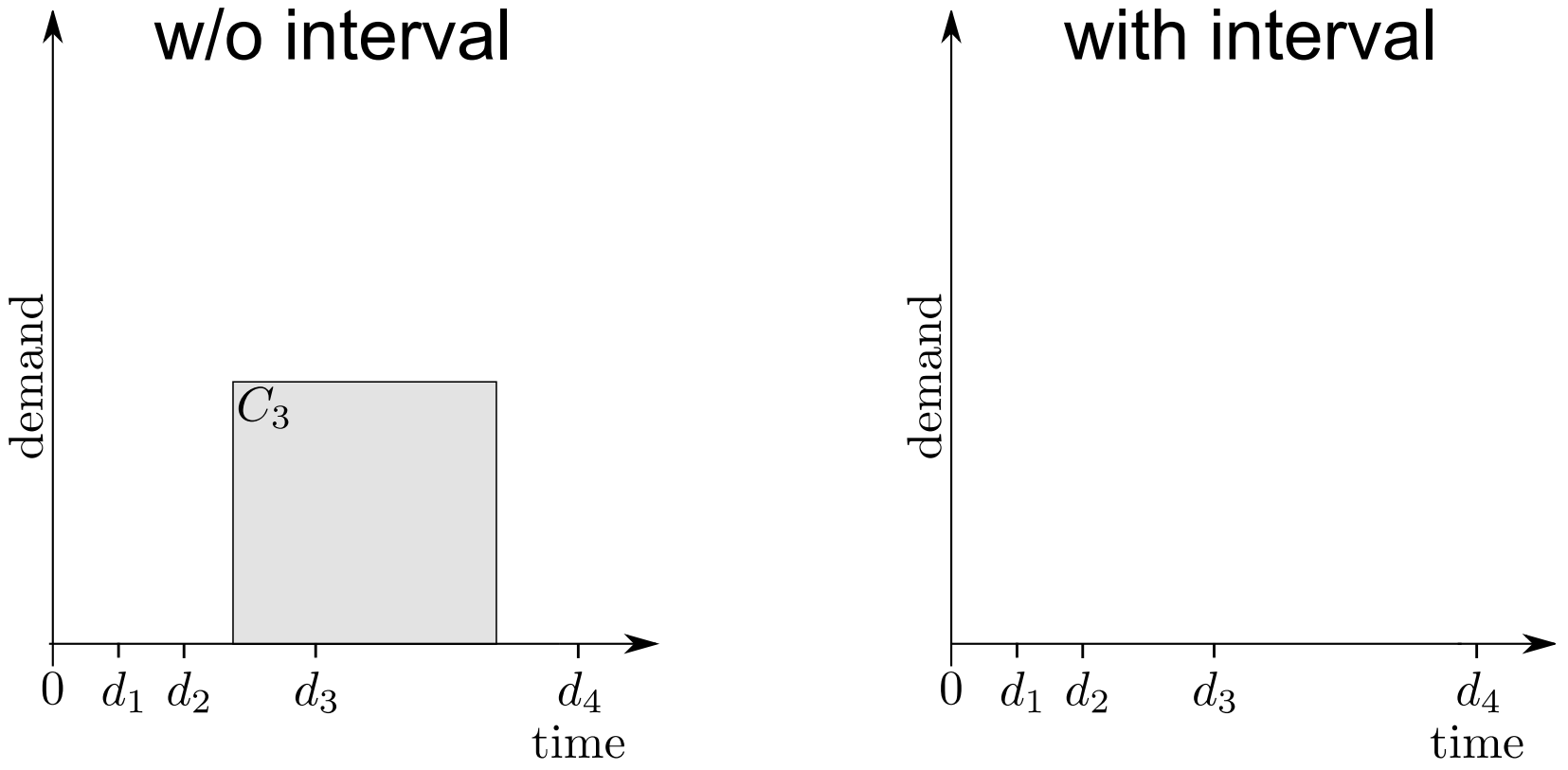
- $\sum price_{ON}(\mathcal{C}_i) \leq i \cdot price(\mathcal{C}_i)$

See paper for inductive proof

$\rightarrow$ ON2D is *k*-competitive, where *k* is the total number of contracts

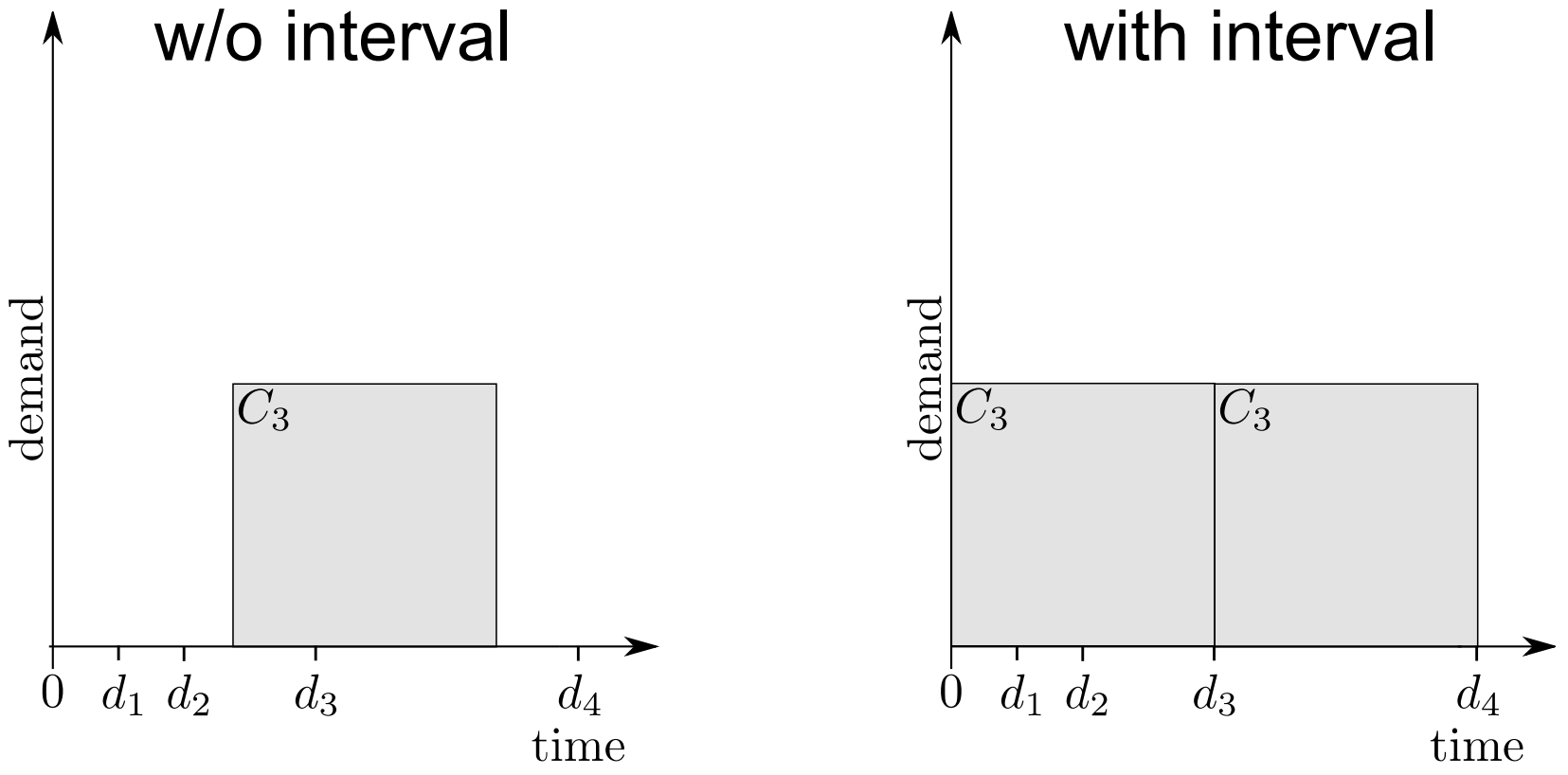
Upper bound relaxing intervals assumption

- ON2D is 2*k*-competitive for the general PPP^2 without *intervals*



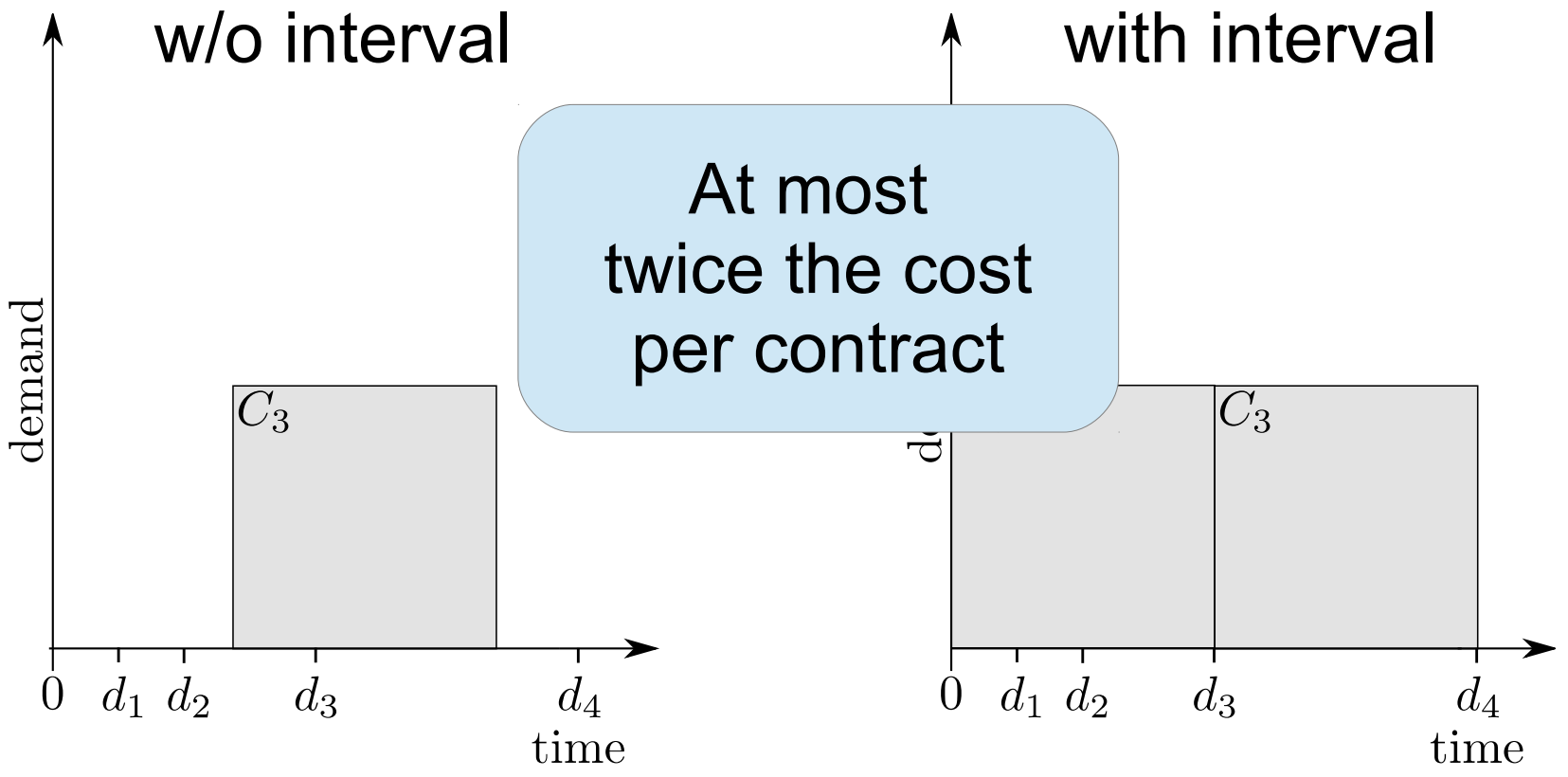
Upper bound relaxing intervals assumption

- ON2D is 2*k*-competitive for the general PPP^2 without *intervals*



Upper bound relaxing intervals assumption

- ON2D is 2*k*-competitive for the general PPP^2 without *intervals*



Lower Bound

- Scenario: each contract is $2k$ times longer and has $2k$ times more rate; contracts double in cost
- Demand scheduled when ON has no contract
- Minimal cost per interval that starts with demand

$\rightarrow$ Lower bound: $k/3$

Offline Algorithm

Algorithm 2 Pre-computation of matrix M for d_k-length

Input: Demand sequence $\sigma_t, \ldots, \sigma_{t+d_k}$ (over interval $[t, t+d_k]$).

Output: Matrix M.

for $i = 1$ to d_k **do**
  $M[i,i] \leftarrow \sigma_{t+i}$
for $i = 1$ to $d_k - 1$ **do**
  for $j = i+1$ to d_k **do**
    $M[i,j] \leftarrow \max\{M[i,j-1], \sigma_{t+j}\}$
return M

Algorithm 3 Offline Algorithm for d_k-length interval

Input: Precomputed matrix M over interval $[t, t+d_k]$.

Output: Optimal total costs $\texttt{OPT}[i,j,\cdot]$ for all intervals within $[t, t+d_k]$.

Initialize all entries in $\texttt{OPT}$ to be 0.
Let $\hat{\sigma} = M[i,j]$.
for $i = 1$ to d_k **do**
  for $\lambda = M[i,i] - 1$ to 0 **do**
    $\texttt{OPT}[i,i,\lambda] \leftarrow \min_{C(r,d)\in\mathcal{C}}\{\texttt{OPT}[i,i,\min\{\hat{\sigma}, \lambda+r\}] + \pi(r,d)\}$
for $\ell = 2$ to d_k **do**
  for $i = 1$ to $d_k - \ell + 1$ **do**
    $j = i + \ell - 1$
    for $\lambda = M[i,j] - 1$ to 0 **do**
      $\texttt{OPT}[i,j,\lambda] \leftarrow \min_{i\le z<j}\{\texttt{OPT}[i,z,\min\{M[i,z],\lambda\}] + \texttt{OPT}[z+1,j,\min\{M[z+1,j],\lambda\}]\}$
      $\mathcal{C}' \leftarrow \{C^{(x)}(t^{(x)}, r, d) \in \mathcal{C} : t^{(x)} = b \cdot d$ for some positive integer b and $t^{(x)} \le i < j \le t^{(x)} + d\}$
      if $\mathcal{C}'$ is not empty **then**
        $\texttt{OPT}[i,j,\lambda] \leftarrow \min\{\texttt{OPT}[i,j,\lambda]; \min_{C(r,d)\in\mathcal{C}'} \texttt{OPT}[i,j,\min\{\hat{\sigma}, \lambda + r\}] + \pi(r,d)\}$
return $\texttt{OPT}[1, d_k, 0]$

Offline Algorithm

Algorithm 2 Pre-computation of matrix M for d_k-length

Input: Demand sequence $\sigma_t, \ldots, \sigma_{t+d_k}$ (over interval $[t, t+d_k]$).

Output: Ma

1: **for** $i =$
2: $M[i$
3: **for** $i =$
4: **for**
5: M
6: **return**

Algorithm 3 Offline Algorithm for d_k-length interval

Input: Precomputed matrix M over interval $[t, t+d_k]$.

ll intervals

$\leftarrow$

$(r, d)\}$

$\leftarrow$

$+$

$\cdot\, d$ for

$< j \leq$

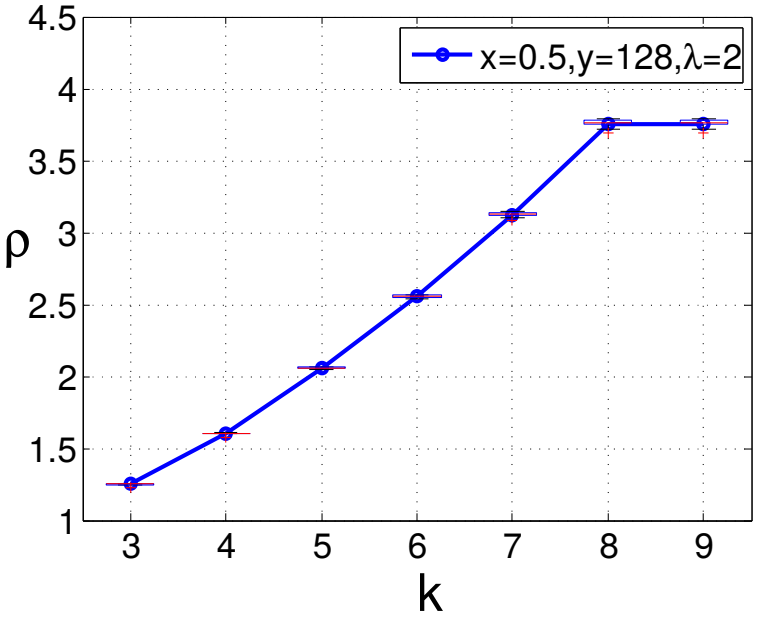
$\{\texttt{OPT}[i, j, \lambda];$

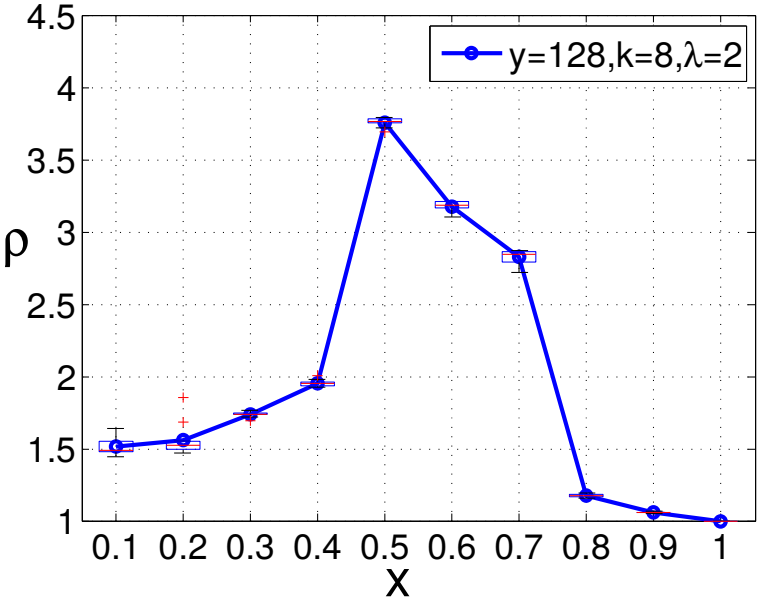
$\min_{C(r,d) \in \mathcal{C}'} \texttt{OPT}[i, j, \min\{\hat{\sigma}, \lambda + r\}] + \pi(r, d)\}$

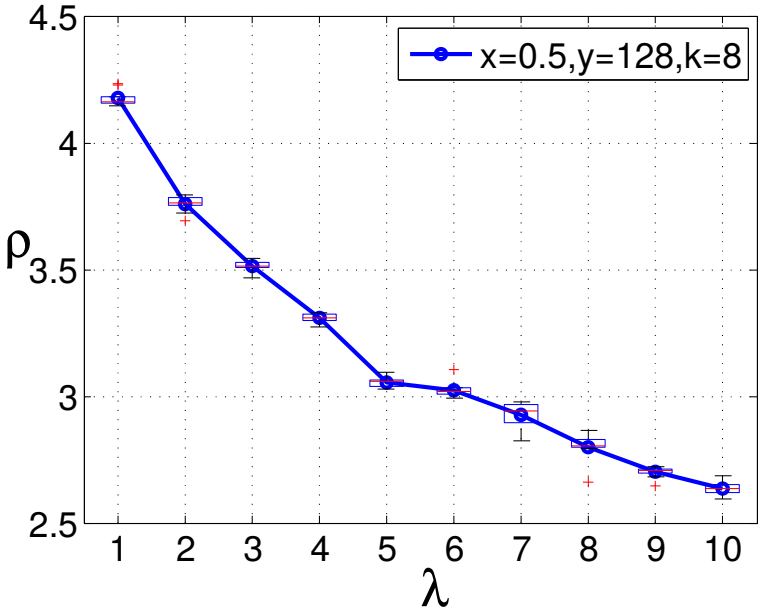
14: **return** $\texttt{OPT}[1, d_k, 0]$

- Dynamic program:
  - Split time into intervals
  - Compute minimum per interval
- Polynomial runtime

Simulation verification







Conclusion

- Introduction of PPP²
- Deterministic online algorithm $O(k)$
- Almost optimal (lower bound $k/3$)
- Polynomial offline algorithm
- Algorithms and results generalize to higher dimensions