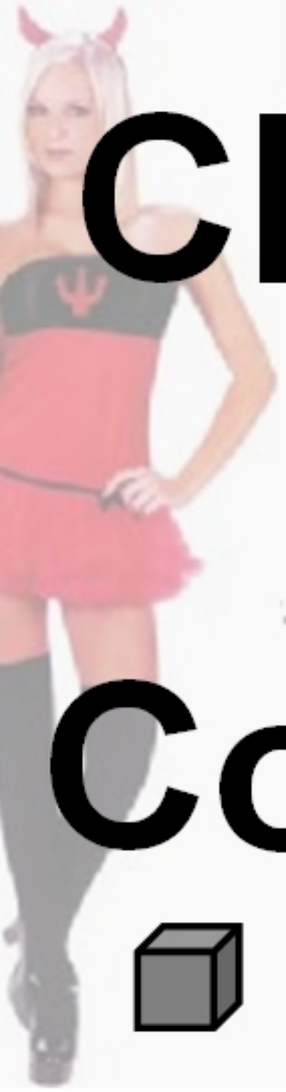
Scheduling of Independent Tasks



Clairvoyance or Cooperation?

Parallelisation of rendering algorithms

University of Paderborn

University of BRISTOL

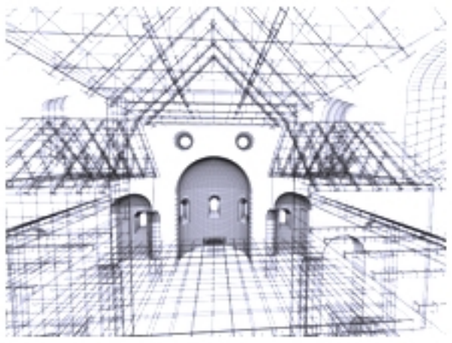
Comenius Uni Bratislava

Project HiQoS (Uni Paderborn et al): parallel ray tracing & radiosity

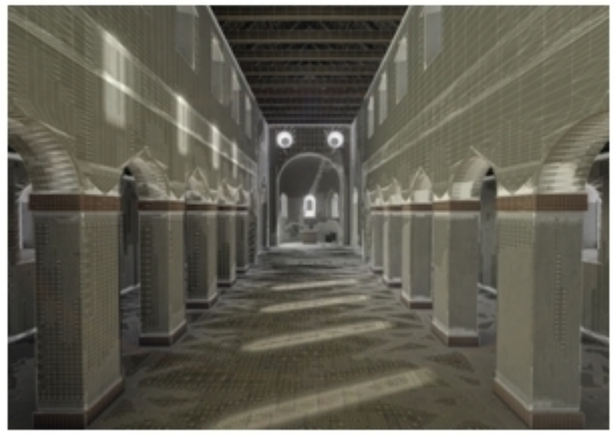
Ray tracing:
„method for everybody“

hour

Radiosity:
„method for professionals“



... many hours



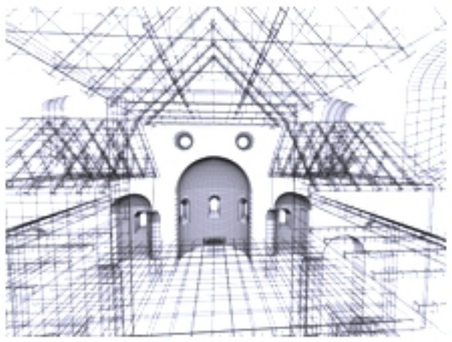
sec

Project HiQoS (Uni Paderborn et al): parallel ray tracing & radiosity

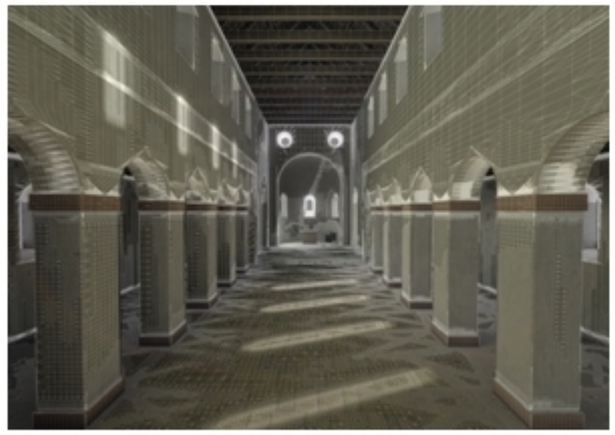
Ray tracing:
„method for everybody“

minutes

Radiosity:
„method for professionals“



... many
minutes

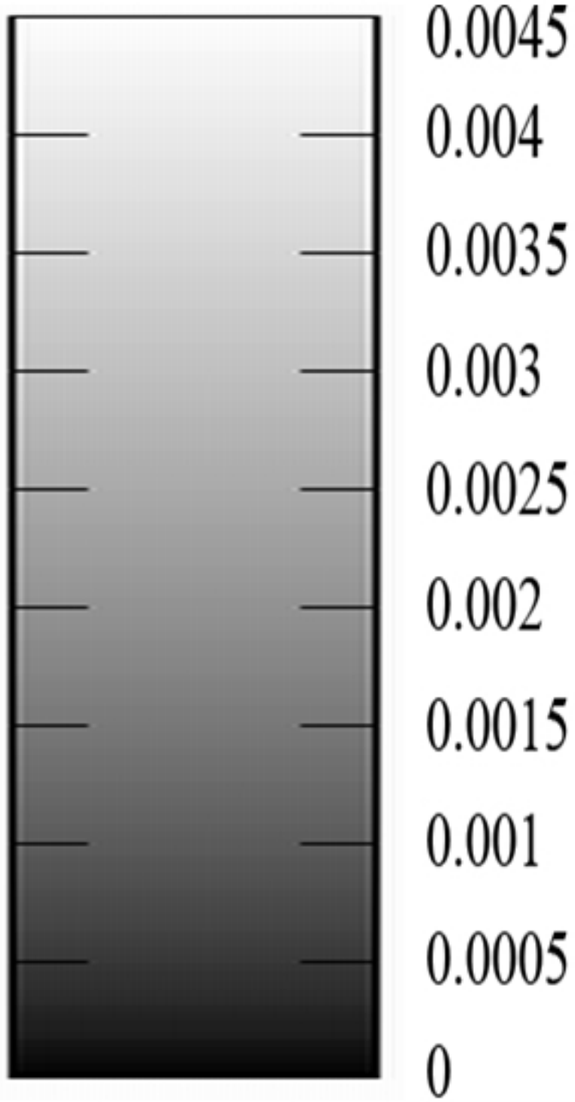


sec

Background: parallel ray tracing (how to distribute the work?)

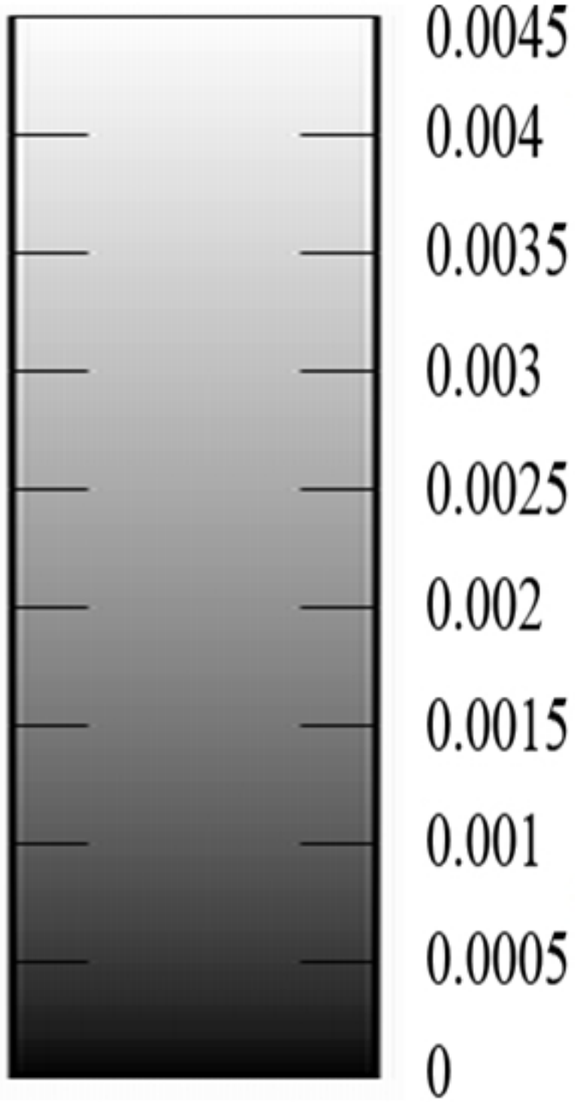
Background: parallel ray tracing (how to distribute the work?)

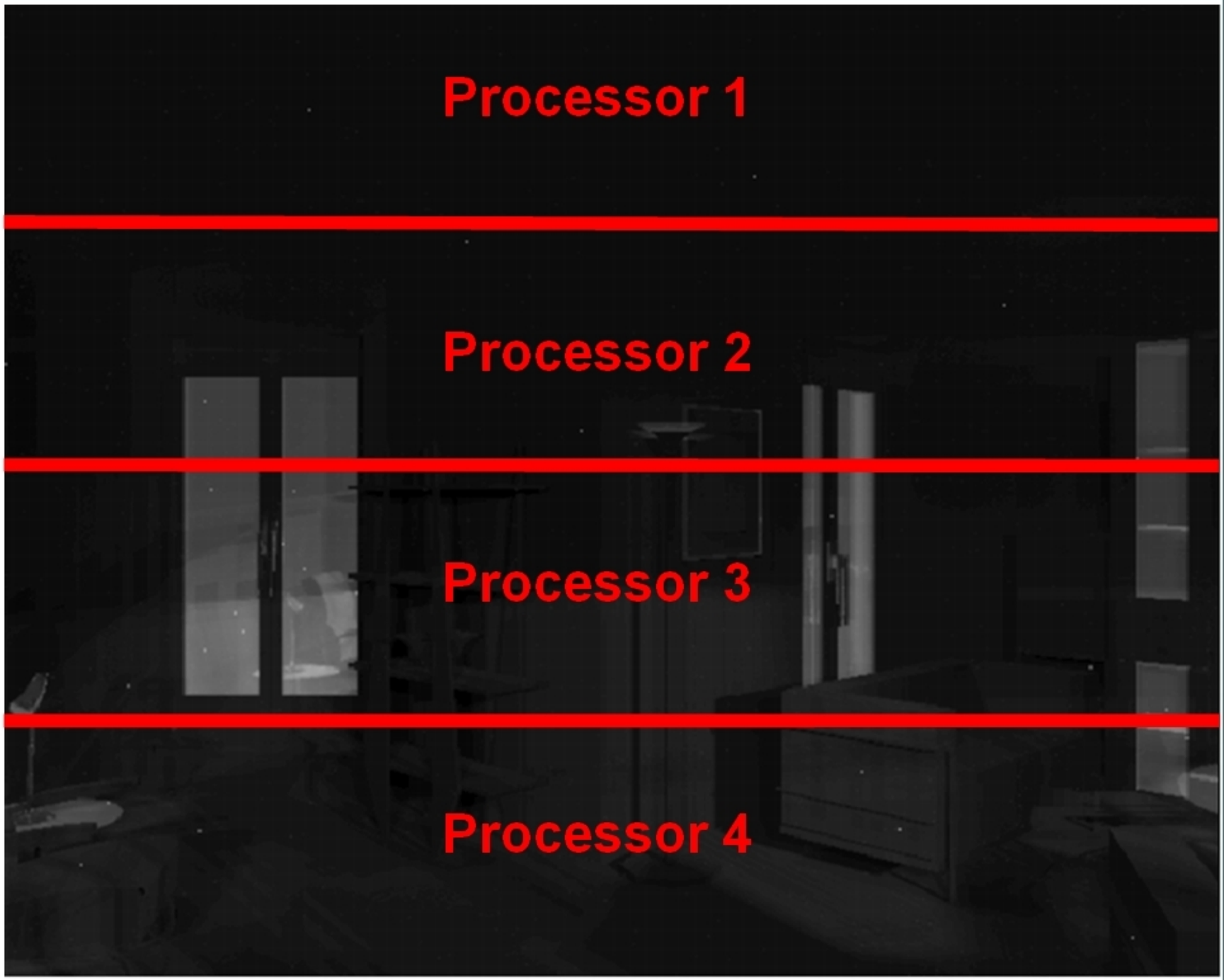
Computational times of pixels



Background: parallel ray tracing (how to distribute the work?)

Computational times of pixels





Although work distribution is equal, time is not

Clairvoyance vs. Cooperation in Scheduling of Independent Tasks

jail
(limits someone's freedom)

tasks
(differ only in contents)

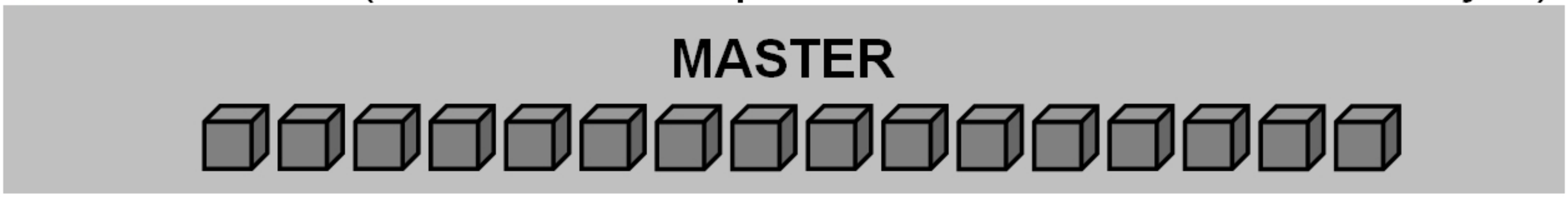
bad

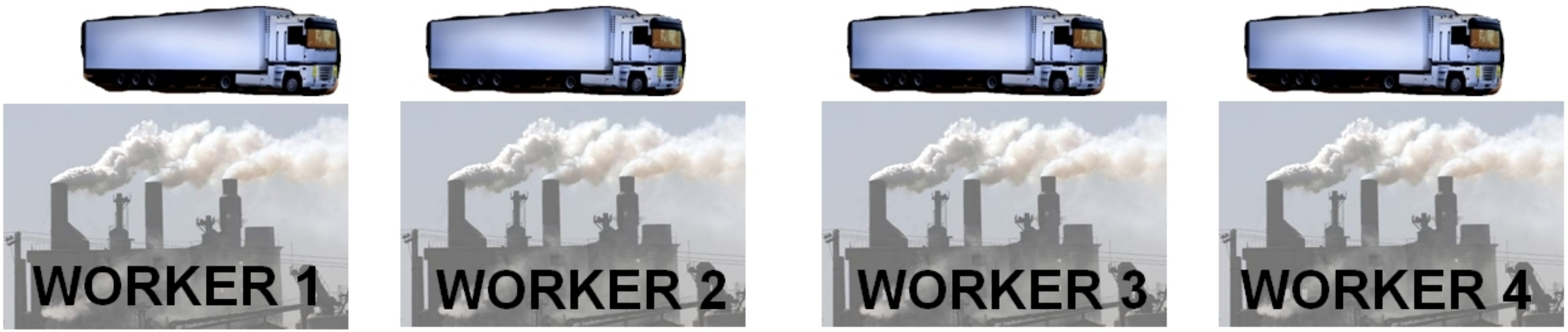
Introduction

W = 16 tasks

N = 4 workers

L = ? seconds (a constant independent on the nr of tasks in a job)



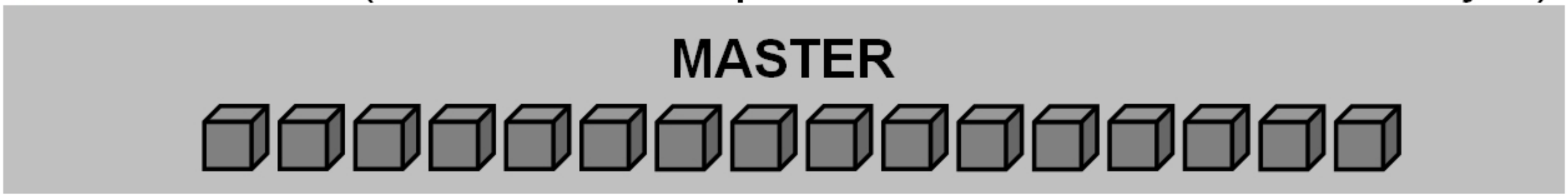


Introduction

W = 16 tasks

N = 4 workers

L = ? seconds (a constant independent on the nr of tasks in a job)

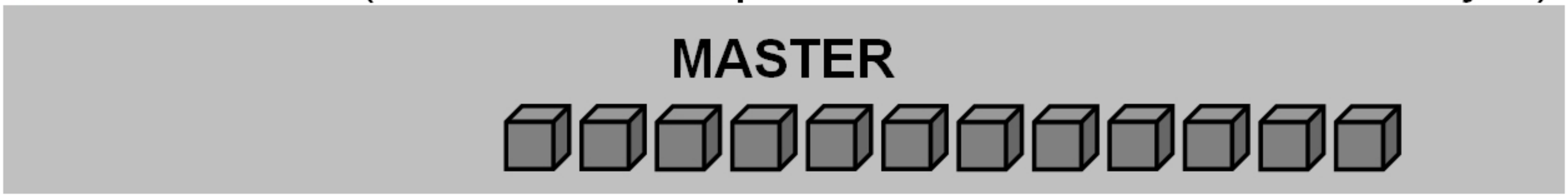


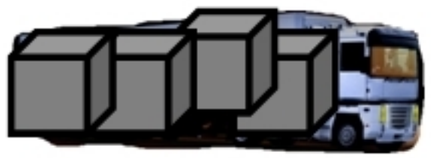
Introduction

W = 16 tasks

N = 4 workers

L = ? seconds (a constant independent on the nr of tasks in a job)



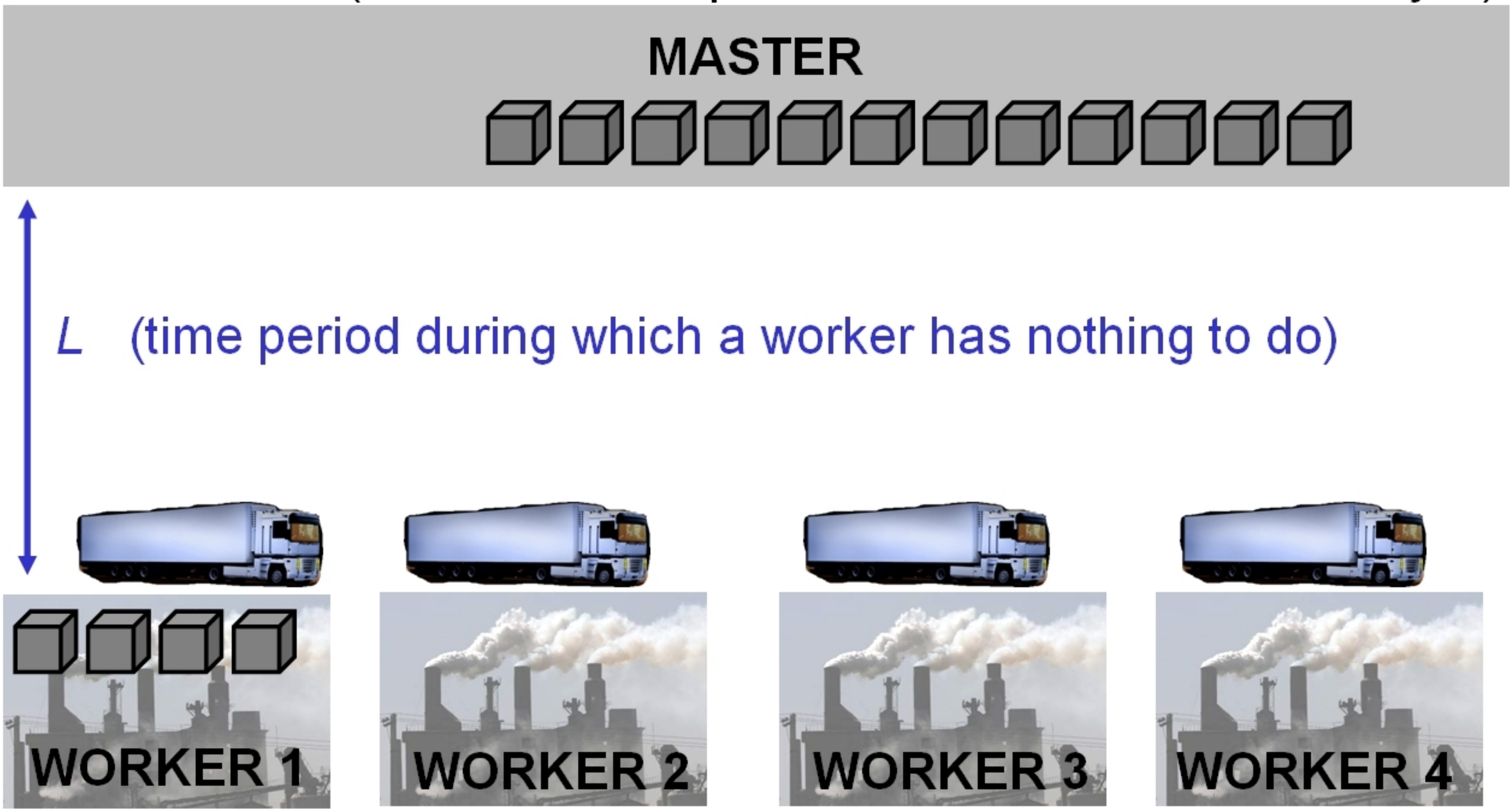


Introduction

W = 16 tasks

N = 4 workers

L = ? seconds (a constant independent on the nr of tasks in a job)



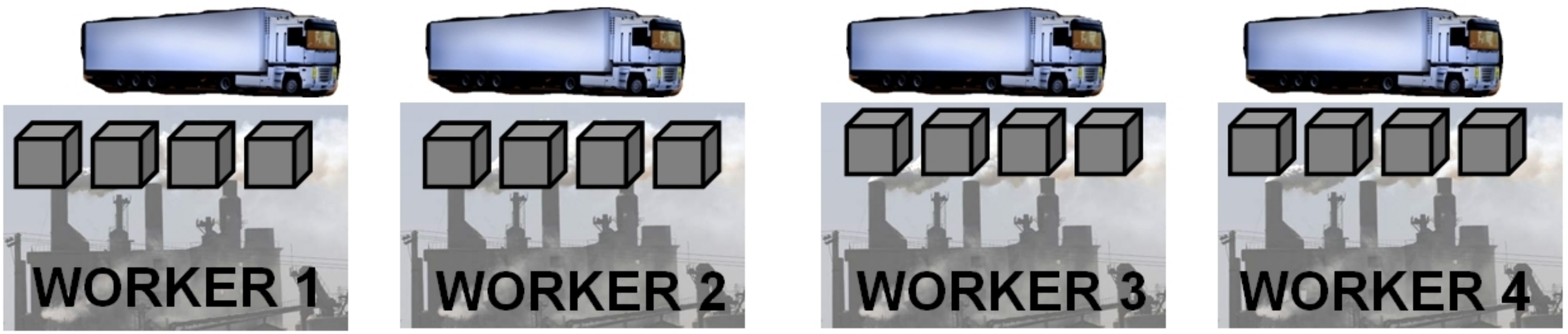
Introduction

W = 16 tasks

N = 4 workers

L = ? seconds (a constant independent on the nr of tasks in a job)

MASTER



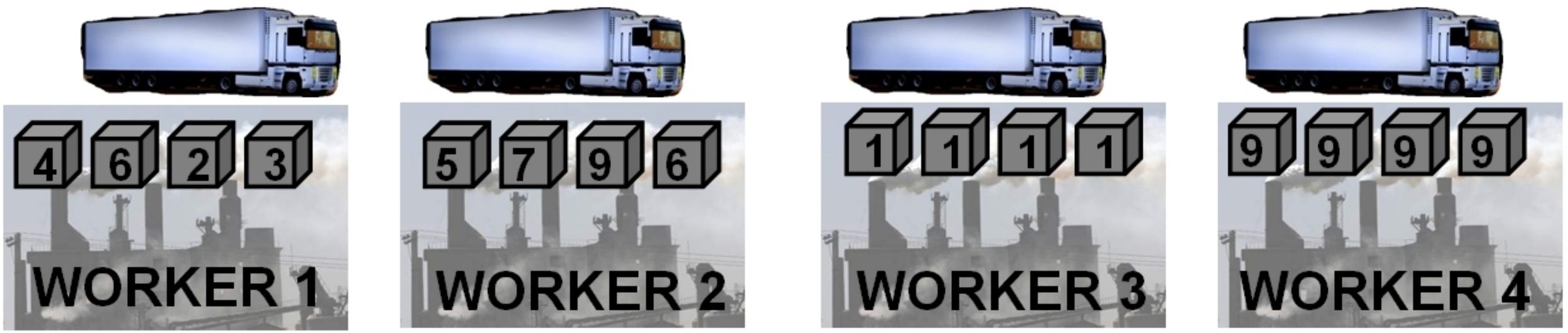
Introduction

W = 16 tasks

N = 4 workers

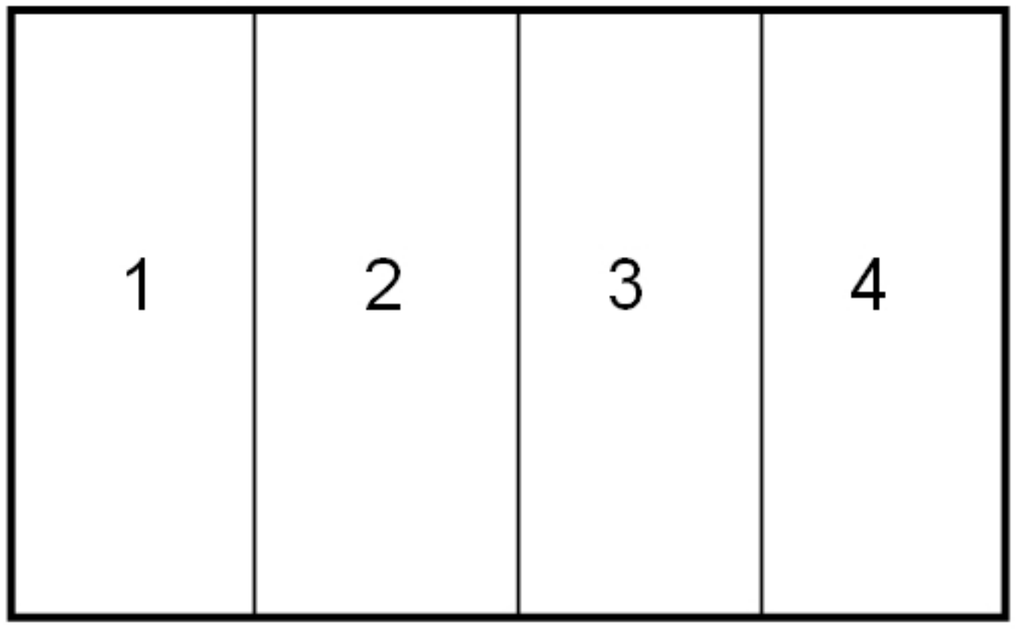
L = ? seconds (a constant independent on the nr of tasks in a job)

MASTER

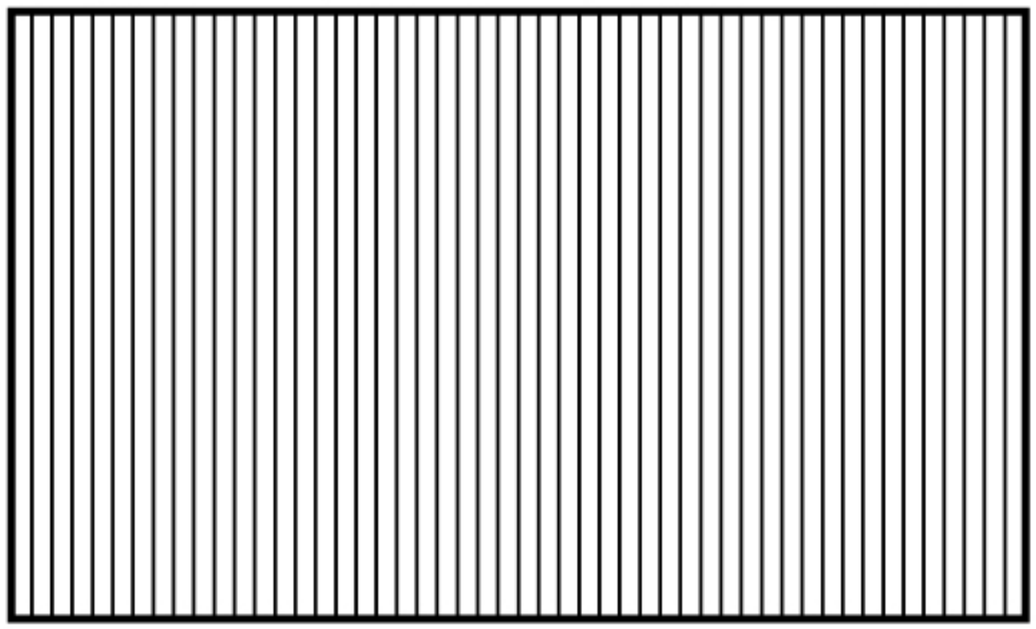


Chunking algorithm (goal: minimise makespan)

Static job assignment



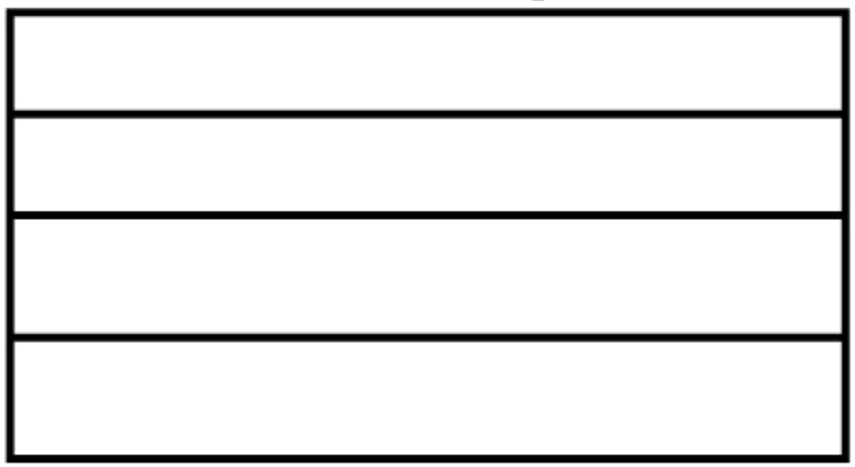
Dynamic job assignment



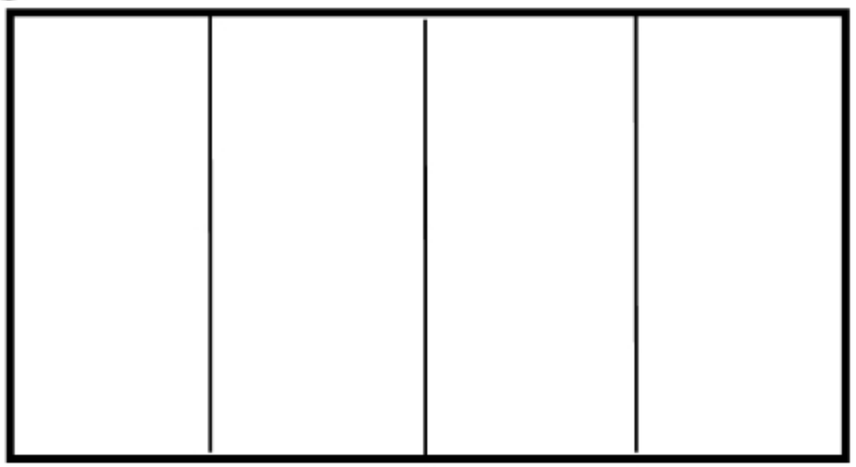
- almost no communication
- heavy imbalance

- heavy communication
- perfect balance (up to 1 task)

Job "shape" is irrelevant, only job size matters



≡



MASTER process

```
master(int W, int N)
{
  int K;
  int work = W;
  while (work > 0)
  {
    wait for a job request from an idle worker;
    compute job size K; /* K may be different for different requests */
    assign job of K tasks to the idle worker;
    work = work - K;
  }
  reply remaining job requests with NO_MORE_WORK;
}
```

***How many tasks* should the MASTER process assign in one *job*? Some practitioners suggest 16 tasks, other practitioners suggest 4096 tasks. Theoreticians produce even more suggestions. (Show must go on.)**

Problem definition

Unknown:
Tasks' durations t_1, t_2, …, t_W
(and also the latency L, if we trust theoreticians)

Given:

N	nr. of worker processes, all equally fast
W	nr. of tasks, independent on each other
L	latency (this is actually also unknown, but perhaps we trust network providers)

CLAIRVOYANCE: partial information on tasks' durations

Clairvoyant probabilistic model:
e.g. tasks' durations are generated by $N(\mu, \delta)$ and μ, δ are a-priori known

Goal:
Minimise expected time

Clairvoyant deterministic model:
e.g. maximal task's duration T_{max} is a-priori known,
or $T = T_{max} / T_{min}$ is a-priori known
or…

Goal:
Minimise maximal time (makespan)

Clairvoyant deterministic model

Unknown:

Tasks' durations t_1, t_2, …, t_W

Given:

N	nr. of worker processes, all equally fast
W	nr. of tasks, independent on each other
L	latency (assignment of one job)

and some clairvoyant information on tasks' durations,
e.g. $T_{max} = \max(t_1,\ldots,t_W)$

Goal:

Minimise maximal time ("worst case"). Mmm, what is "worst case"?

Chunking algorithm (goal: minimise maximal makespan)

Given:

N nr. of workers

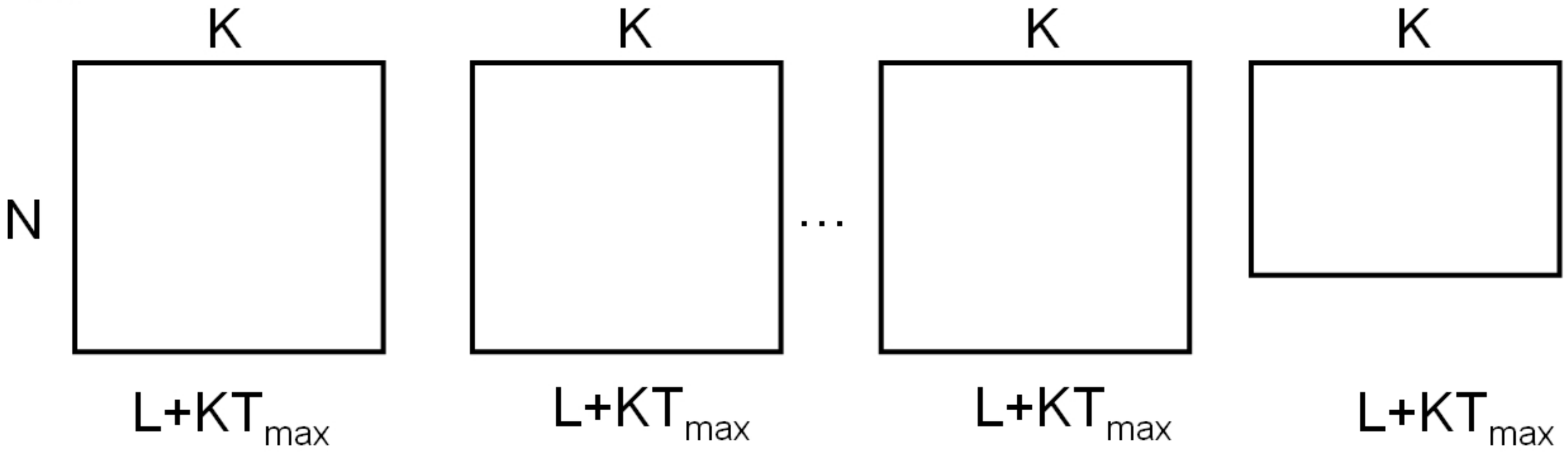
W nr. of tasks

L latency

T_{max} max. task complexity

Unknown:

K_{opt} (chunk size)



$$M = \left(1 + \frac{W}{NK}\right)(L + KT_{\max}) \qquad K_{opt} = \sqrt{\frac{WL}{NT_{\max}}} \rightarrow M_{opt} = \frac{WT_{\max}}{N} + L + 2\sqrt{\frac{WT_{\max}L}{N}}$$

$$M'(K) := 0$$

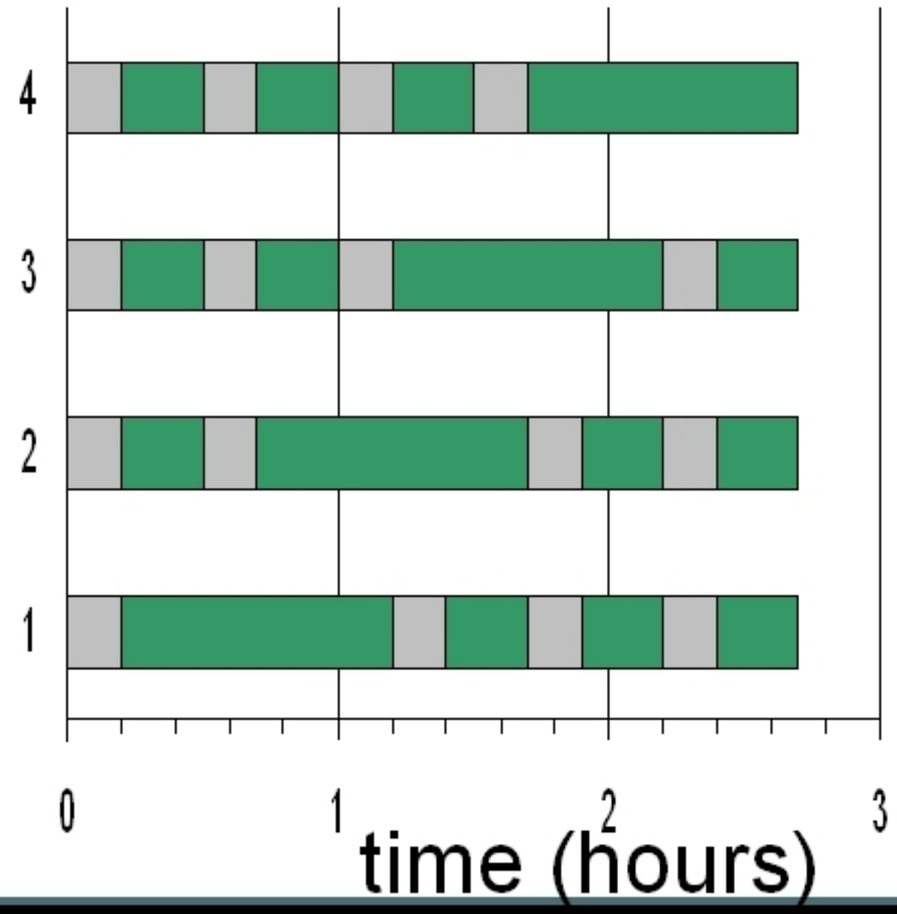
Minimise maximal time? Worst case depends on input!

Goal:

Minimise maximal time (in “worst case”)

Example: $N = 4$, $W = 16$, $L = 12$ min, $T_{max} = 1$ hour. Let us use chunking algorithm which always assigns jobs of the same size K. The “worst case” is that all W tasks are of duration $T_{max} = 1$ hour (indeed, this maximises the time). We want to set K so that we minimise the makespan (total time)

“K=4 is the best choice”



“Come on. What if only 4 tasks are long? K=1 is a better choice”

Clairvoyant deterministic model, revised

Unknown:
Tasks' durations $t_1, t_2, \ldots, t_W$

Given:

N	nr. of worker processes, all equally fast
W	nr. of tasks, independent on each other
L	latency (assignment of one job)

and some clairvoyant information on tasks' durations, e.g. T_{max}

Goal:
Minimise competitive ratio 

Definition. **Competitive ratio $CR_{S(A)}(W, N, L)$** of algorithm S with a-priori information A is the maximal ratio between the makespan of algorithm S and the makespan of the best offline algorithm—over all sequences $t_1, \ldots, t_W$ which conform to the a-priori information A:

$$\mathcal{CR}_{S(A)}(W, N, L) = \sup_{t_1 \ldots t_W} \frac{\mathcal{M}_{S(A)}(t_1 \ldots t_W)}{\mathcal{M}_{\text{best\_offline}}(t_1 \ldots t_W)}$$

Definition. An algorithm with a **smaller competitive ratio is better** than an algorithm with a higher competitive ratio

Notes on competitiveness

$$\mathcal{CR}_{S(A)}(W,N,L) = \sup_{t_1 \ldots t_W} \frac{\mathcal{M}_{S(A)}(t_1 \ldots t_W)}{\mathcal{M}_{\text{best\_offline}}(t_1 \ldots t_W)}$$

An offline algorithm knows everything a-priori (before the computation starts)

Finding the shortest schedule offline is an NP-hard problem, so its computation may take a long time when W is large—but this time does not add to $M_{best\_offline(t_1, \ldots, t_W)}$

However, $M_{best\_offline(t_1, \ldots, t_W)}$ includes one-time penalty L, as the pre-computed schedule is assigned to workers once (in parallel)

$CR_{S(A)} \geq 1$ for any algorithm S. Makespan of S also includes at least one assignment penalty L. Moreover, if S only has a partial a-priori information A on durations $t_1, t_2, \ldots, t_W$, it cannot generally produce the shortest schedule

$CR_{S(A)}$ is a function of W, N, L. It is particularly interesting to observe its limit behaviour for $W \to \infty$. This indicates how good the algorithm S is

Chunking algorithm (goal: minimise competitive ratio)

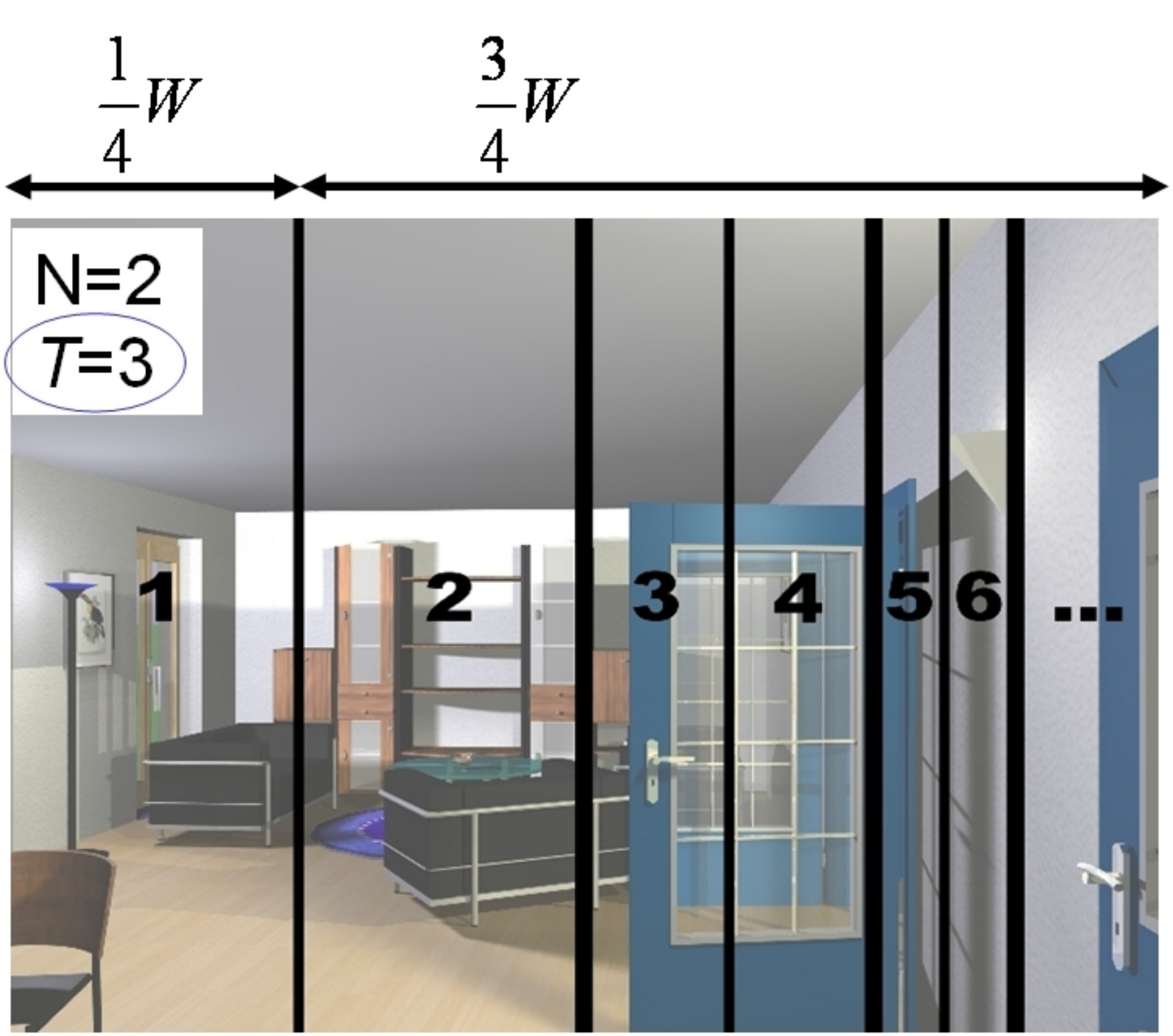
Theorem 1. *For all* W, N, L, T_{max} *such that* $W > N^3 + N^2(N-1)T_{max}/L$ *competitive ratio of an arbitrary assignment algorithm with no work redistribution and with a-priori knowledge of* T_{max} *is at least* N *(i.e.* $\Omega(N)$ *for* $W \to \infty$*).*

👎 CR of chunking increases linearly with an increasing number of workers for $W \to \infty$
(*W* does not actually have to be very large, $W \cong N^3$ will do)

👎 **This holds not only for every setting of *K*, but for all algorithms which know only *W*, *N*, *L* and T_{max} in advance**

👎 **A-priori knowledge of T_{max} does not help much, efficiency of parallel processing goes down the drain**

Factoring algorithm, example

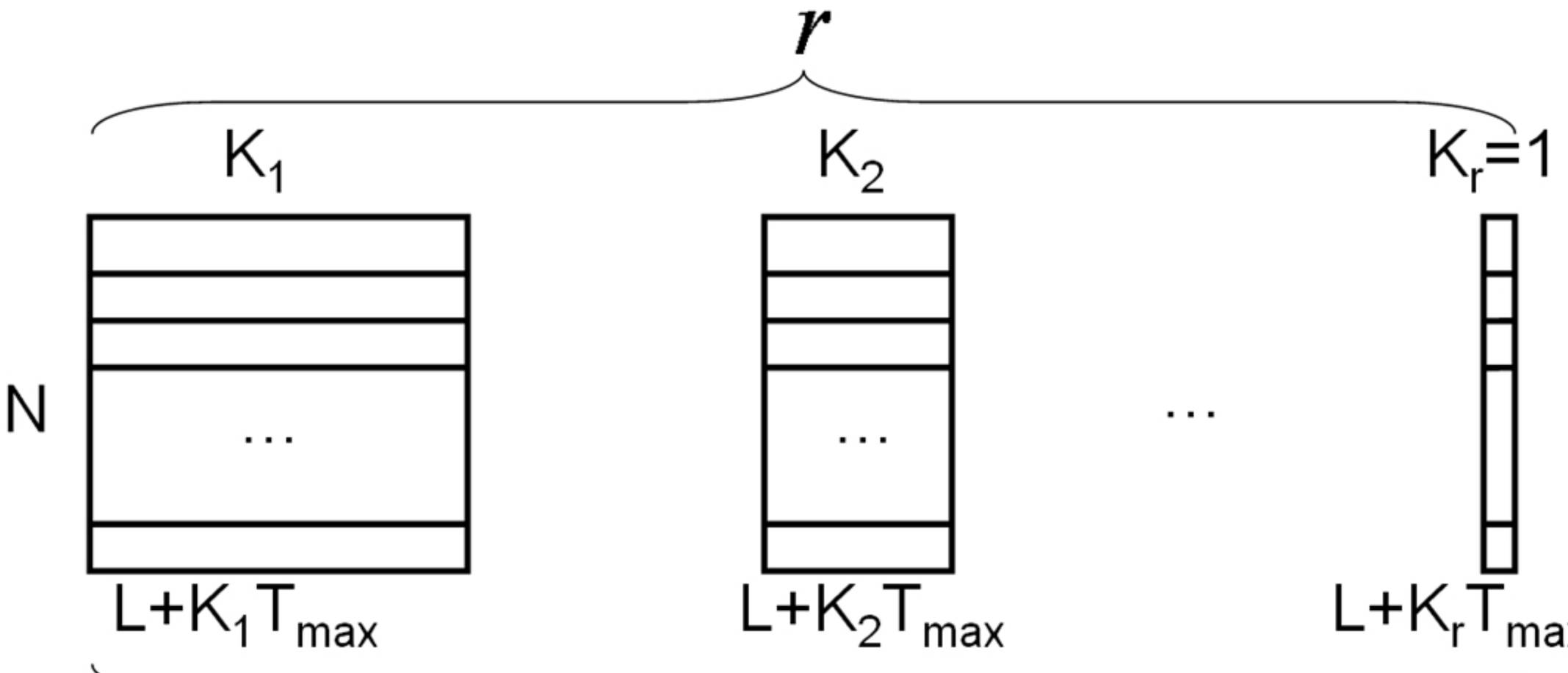


N nr. of workers
W nr. of tasks
T max. ratio of tasks' durations, $T = T_{max} / T_{min}$
Unknown: the job size K_i used in round i

👍 Perfect balance
(max. imbalance is 1 task)

$$K_i = \max\left(1, \left\lfloor \frac{w_i}{1 + T \cdot (N-1)} \right\rfloor\right)$$

Factoring algorithm (goal: minimise competitive ratio)



$$\frac{W}{N}\,T_{\max} + Lr$$

$$K_i = \max\left(1, \left\lfloor \frac{w_i}{1 + T \cdot (N-1)} \right\rfloor\right)$$

This yields $r = \dfrac{\ln(W/N)}{\ln \dfrac{1+T(N-1)}{(N-1)(T-1)}}$

Factoring (goal: minimise competitive ratio)

Theorem 2. *For all N, L, T competitive ratio of the factoring algorithm with a-priori knowledge of T using factor $F = 1 + T(N-1)$ is $1+O((\ln W)/W)$ and approaches 1 for $W \to \infty$.*

👍 This means that the knowledge of $T = T_{max} / T_{min}$ is more valuable than the knowledge of T_{max}

👎 T must be known before it all starts. If T is underestimated or overestimated, then this theorem does not hold

$T \to 1.0 \equiv$ chunking with K = W / N

$T \to \infty \equiv$ chunking with K = 1

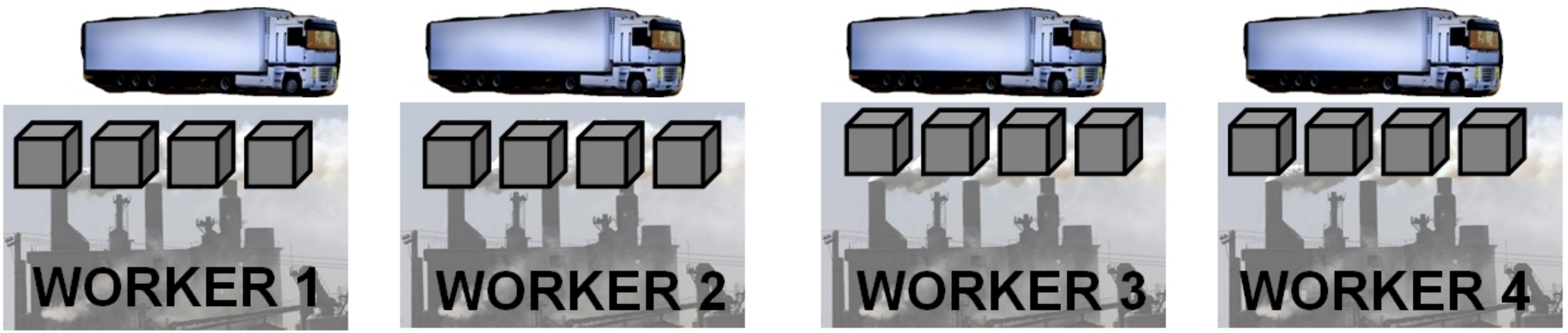
Work stealing algorithm, example

W = 16 tasks

N = 4 workers

L = ? seconds

MASTER



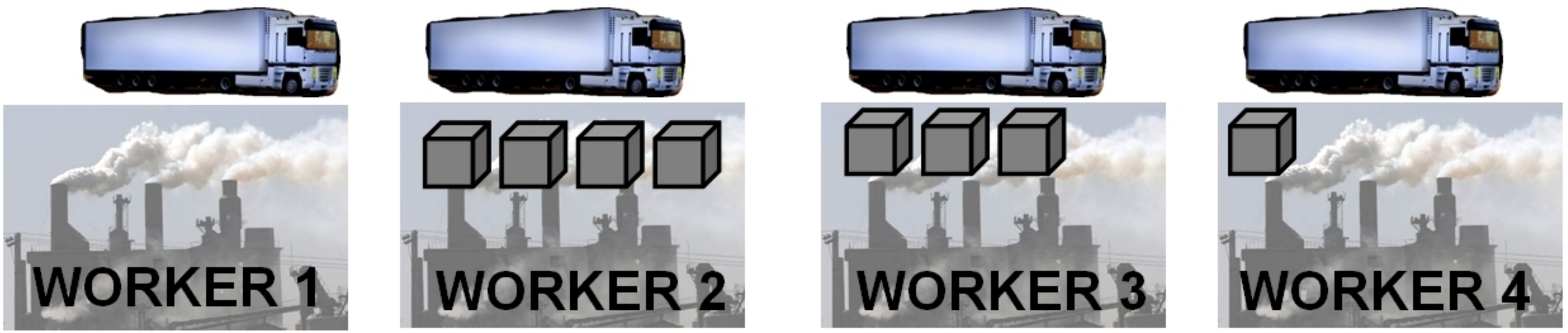
Work stealing algorithm, example

W = 16 tasks

N = 4 workers

L = ? seconds

MASTER



Work stealing algorithm, example

W = 16 tasks

N = 4 workers

L = ? seconds

MASTER

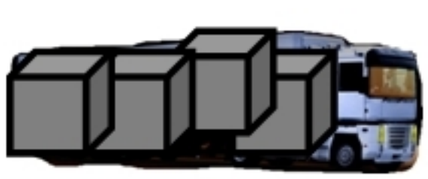
Work stealing algorithm, example

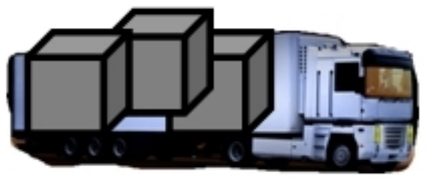
W = 16 tasks

N = 4 workers

L = ? seconds

MASTER





Work stealing algorithm, example

W = 16 tasks

N = 4 workers

L = ? seconds

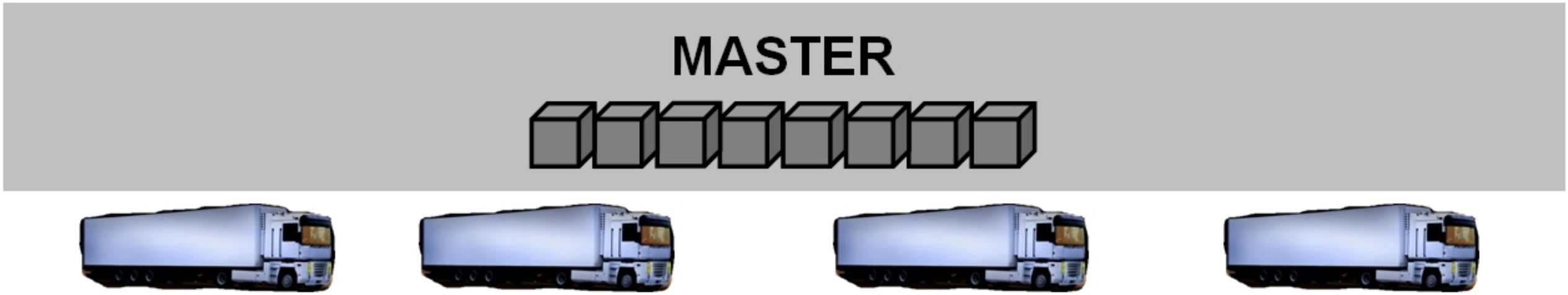
MASTER

Work stealing algorithm, example

W = 16 tasks

N = 4 workers

L = ? seconds



Work stealing algorithm, example

W = 16 tasks

N = 4 workers

L = ? seconds

MASTER

Work stealing algorithm, example

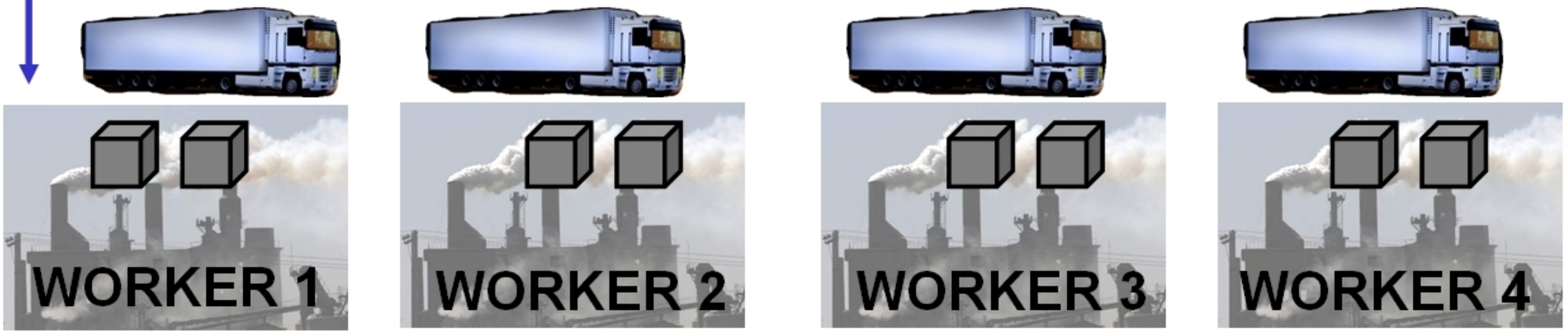
W = 16 tasks

N = 4 workers

L = ? seconds

MASTER

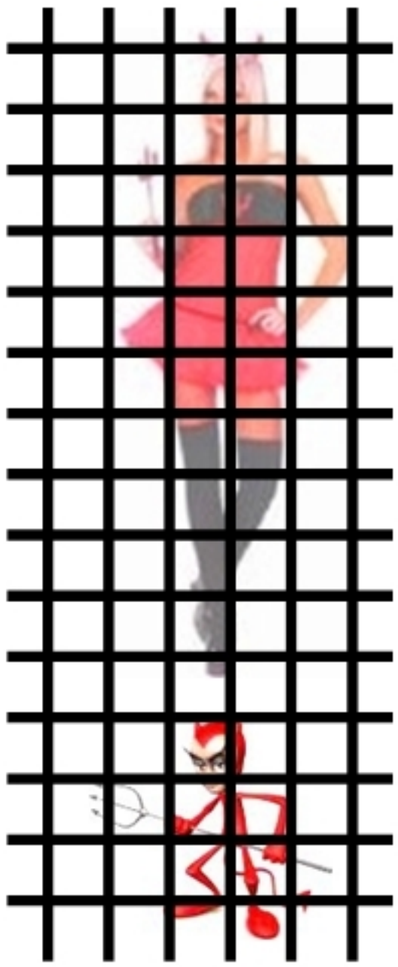
L' (time period during which a worker has nothing to do)



Work stealing algorithm (goal: minimise competitive ratio)

Theorem 3. *For all N, L, L' competitive ratio of the work stealing algorithm with no a-priori knowledge is $1 + O((\ln W)/W)$ and approaches 1 for $W \to \infty$.*

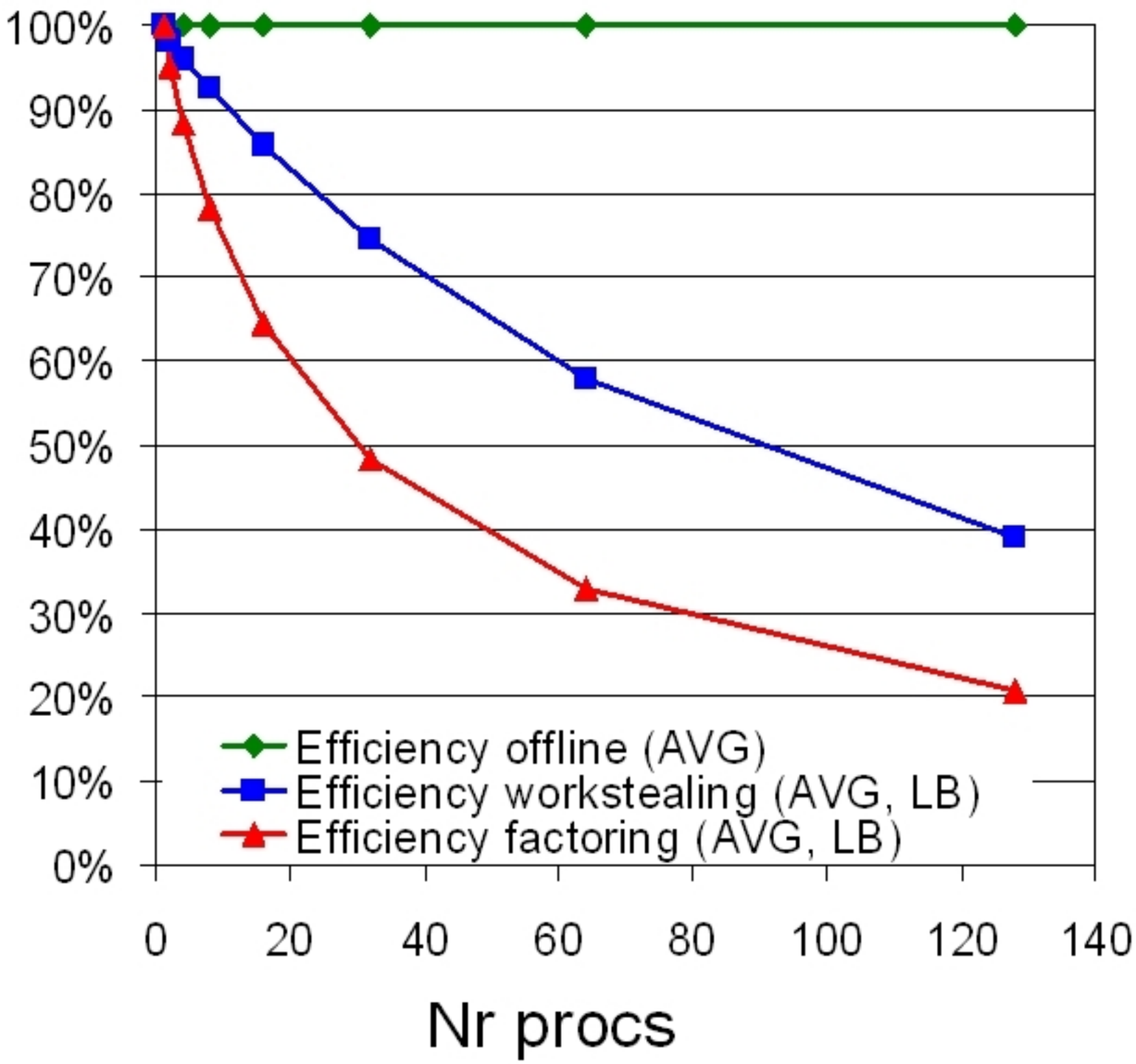
👍 Work stealing has the same asymptotic behaviour as factoring

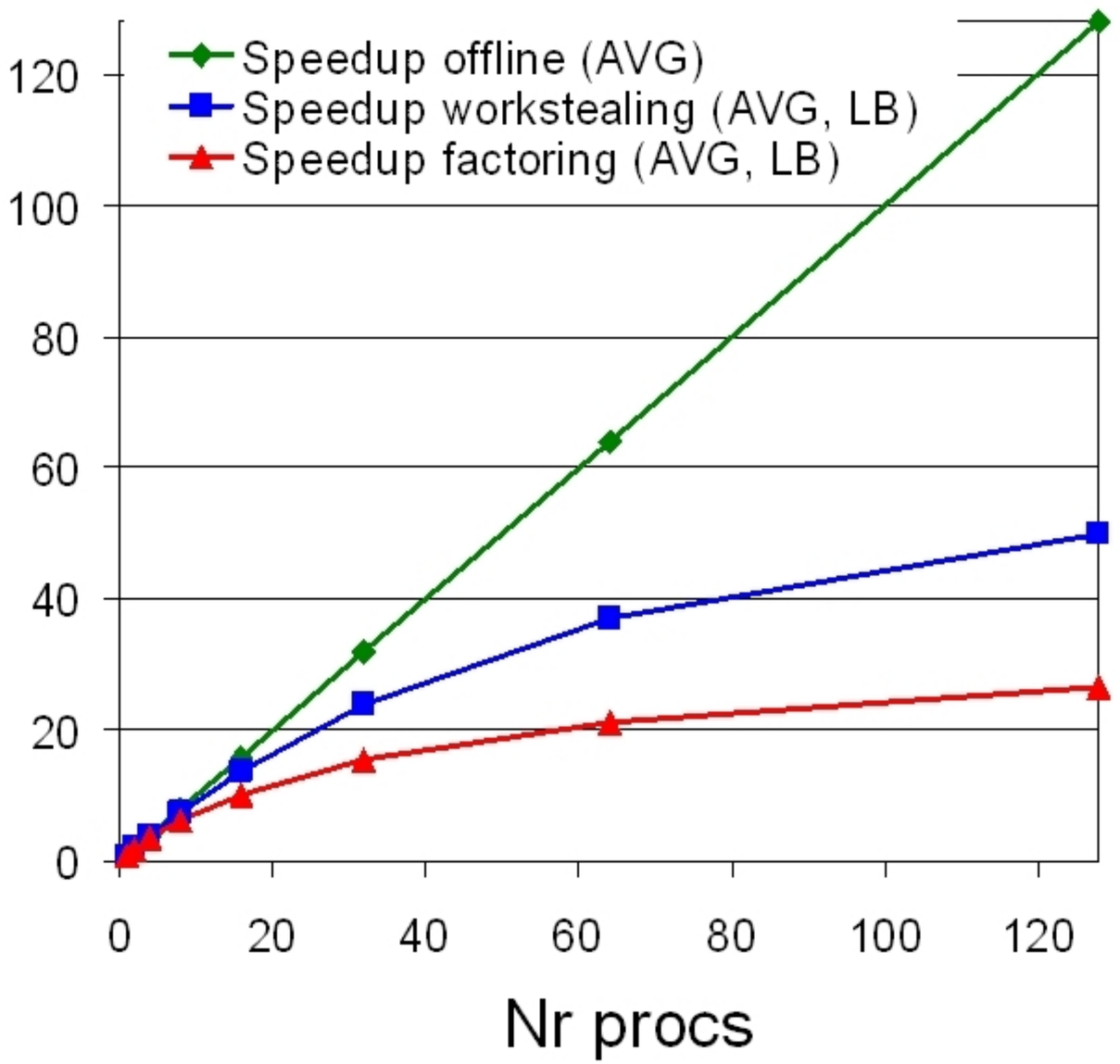


👍 Work stealing needs
NO CLAIRVOYANCE
(i.e. no a-priori information)

Practical consequence: efficient use of resources

L = 0,003 sec
Tmin = 0,01 sec
Tmax = 10 sec
W = 100000





Clairvoyance or Cooperation?

Comparison of 3 online algorithms:

- Chunking, requires knowledge of T_{max}

$lim_{W\to\infty}\ CR = N$, where N is number of workers (i.e. processors)

- Factoring, requires knowledge of T_{max} / T_{min}

$lim_{W\to\infty}\ CR = 1$

- Work stealing, requires no knowledge

$lim_{W\to\infty}\ CR = 1$

Work stealing is a clear winner even in a direct comparison which is not asymptotic—if certain conditions hold (they hold "in practice-relevant settings"). Additionally, work stealing offers attractive extensions: dynamic task creation, fault-tolerance, … Most importantly, **work stealing requires no parameters**

Clairvoyance or Cooperation?