Paul Metzger University of Edinburgh

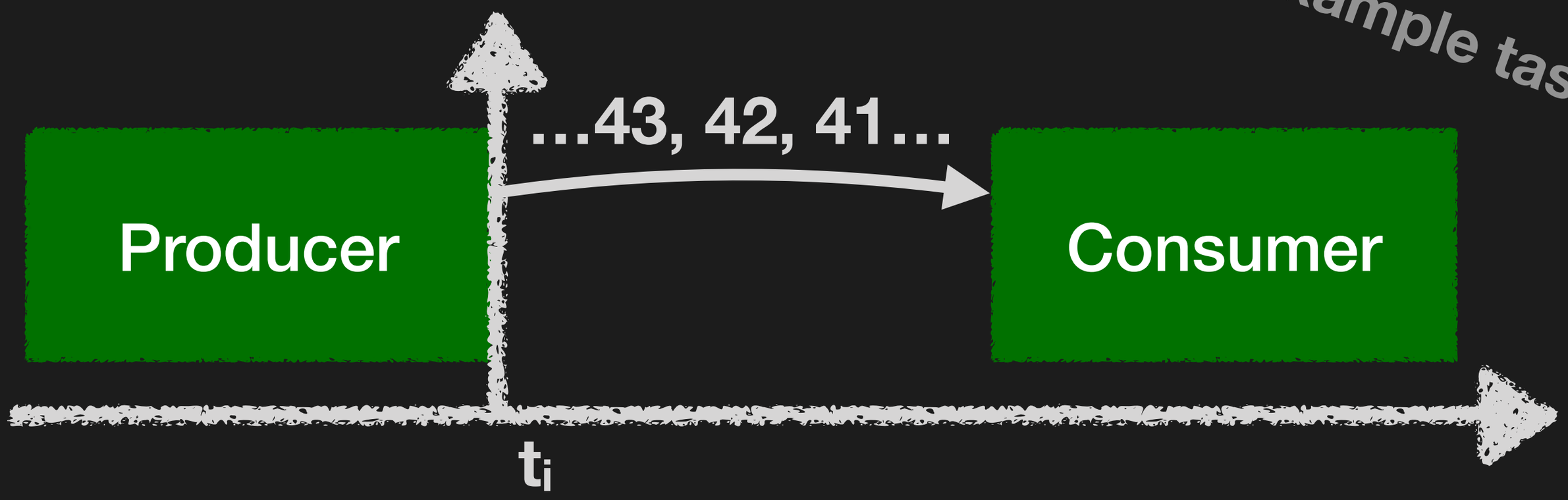
Enforcing Deadlines for Skeleton-based Parallel Programming

rtas2020.paulmetzger.info

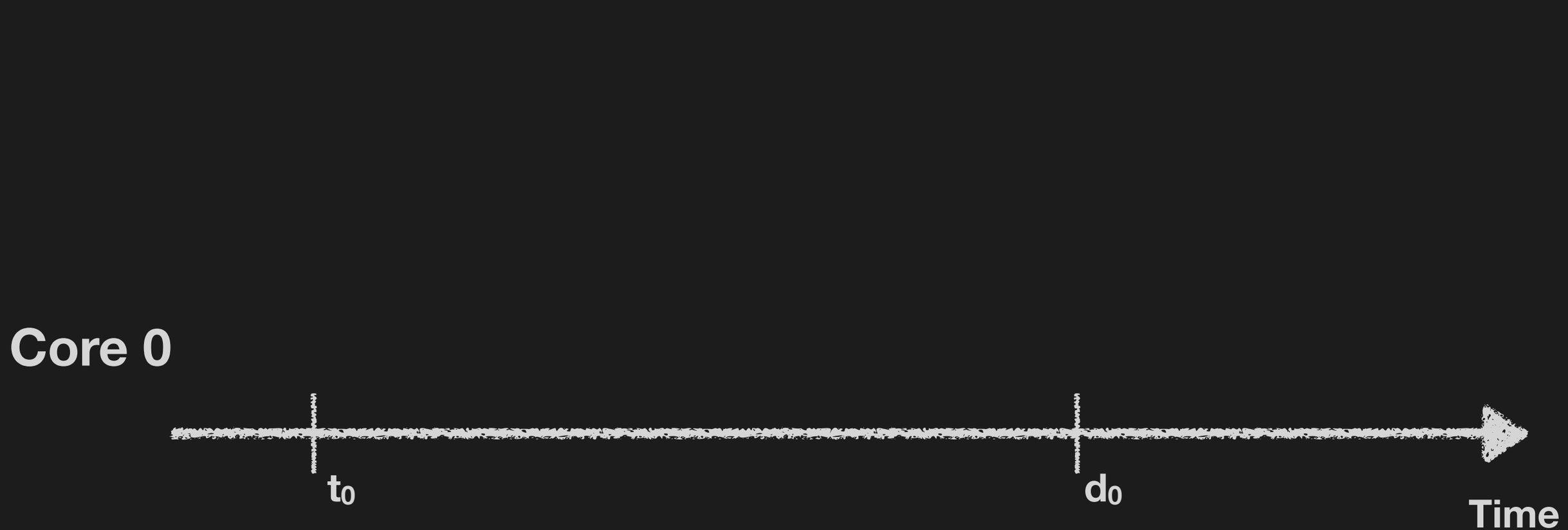
Programming parallel real-time systems is hard

Parallel Real-Time Programming is Hard

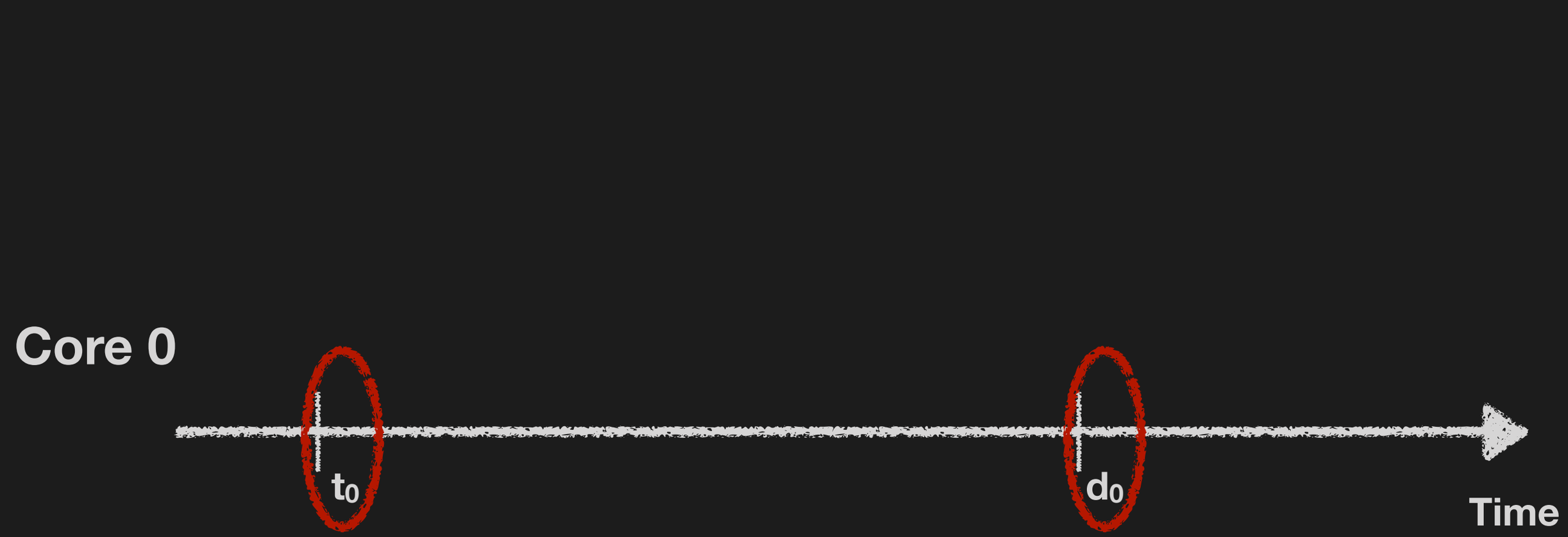
A simple producer-consumer example



Parallel Real-Time Programming is Hard

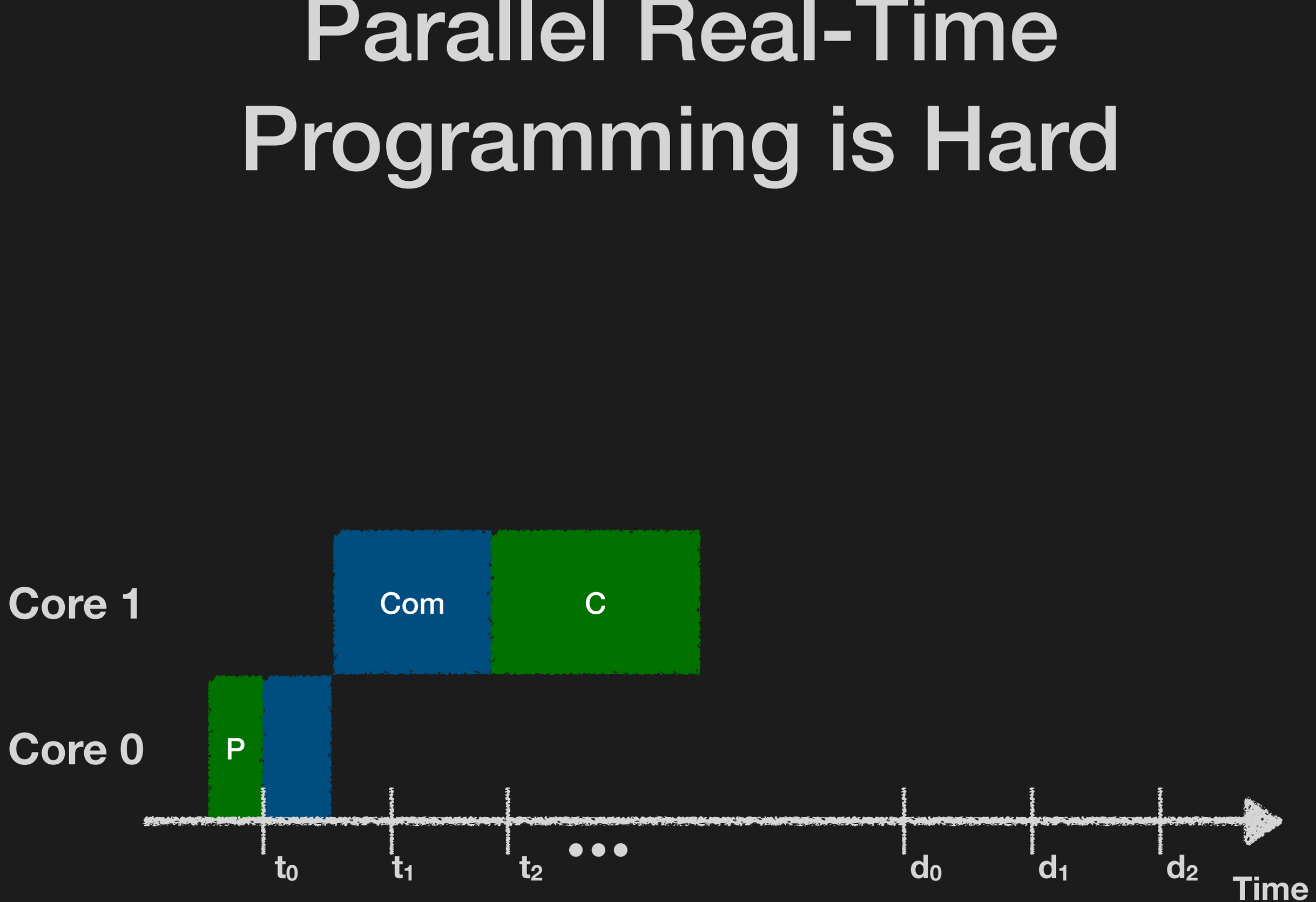


Parallel Real-Time Programming is Hard

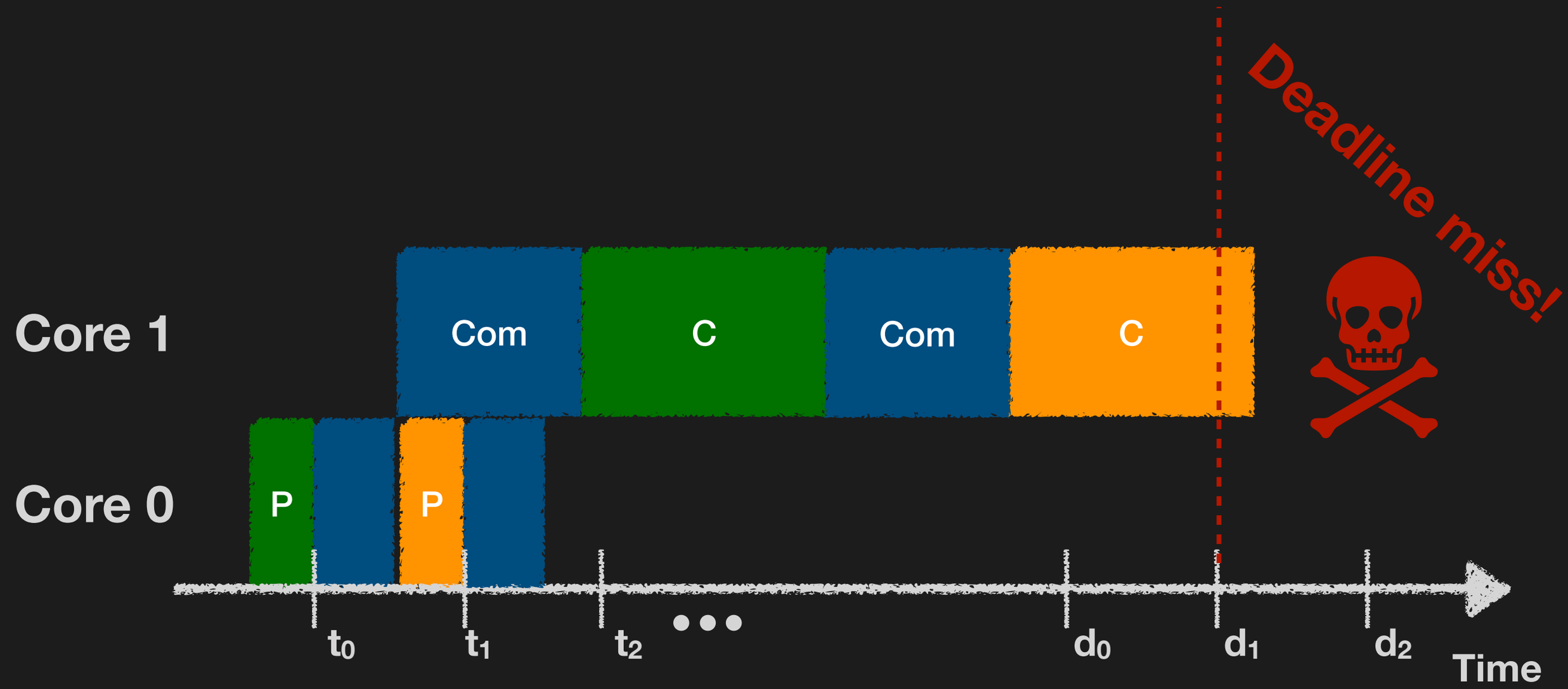


Parallel Real-Time Programming is Hard

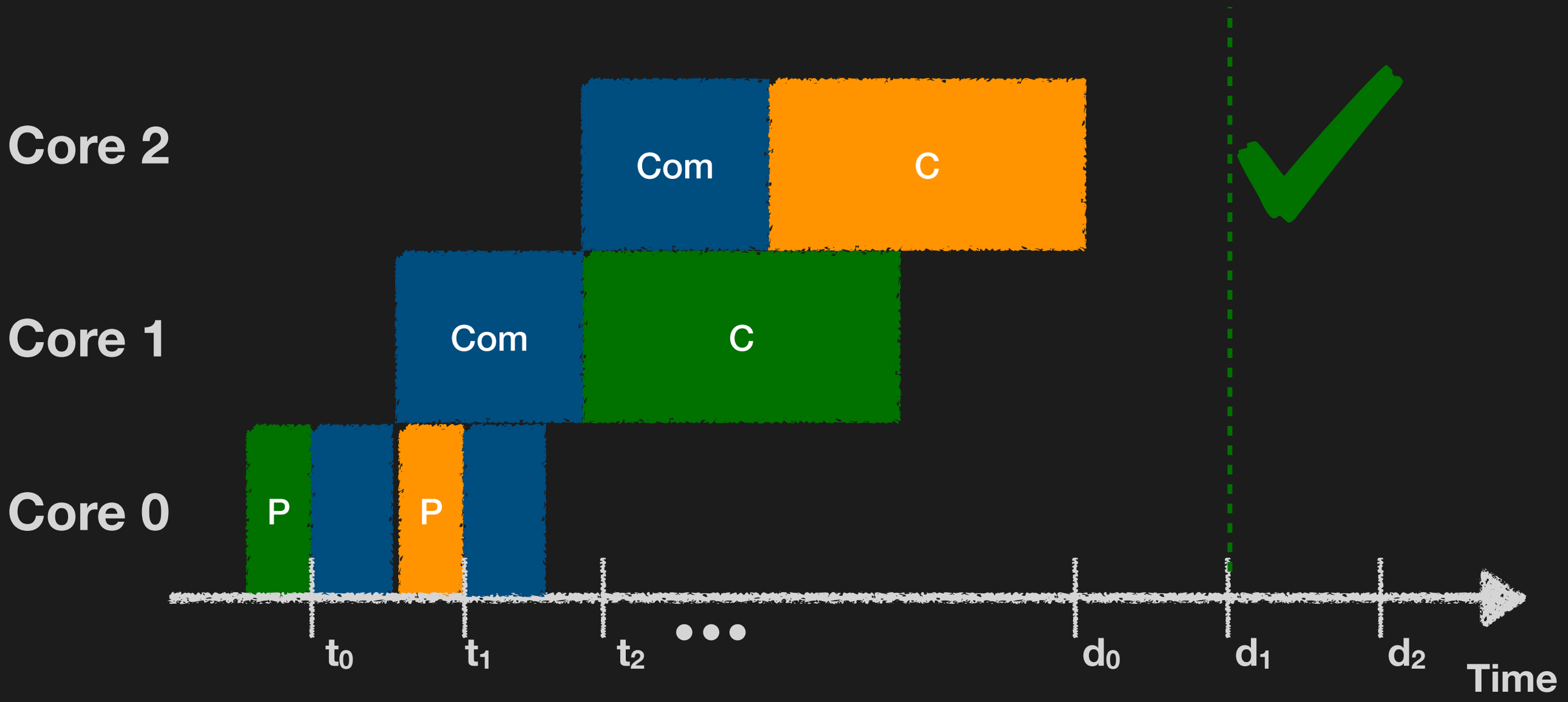
Parallel Real-Time Programming is Hard



Parallel Real-Time Programming is Hard

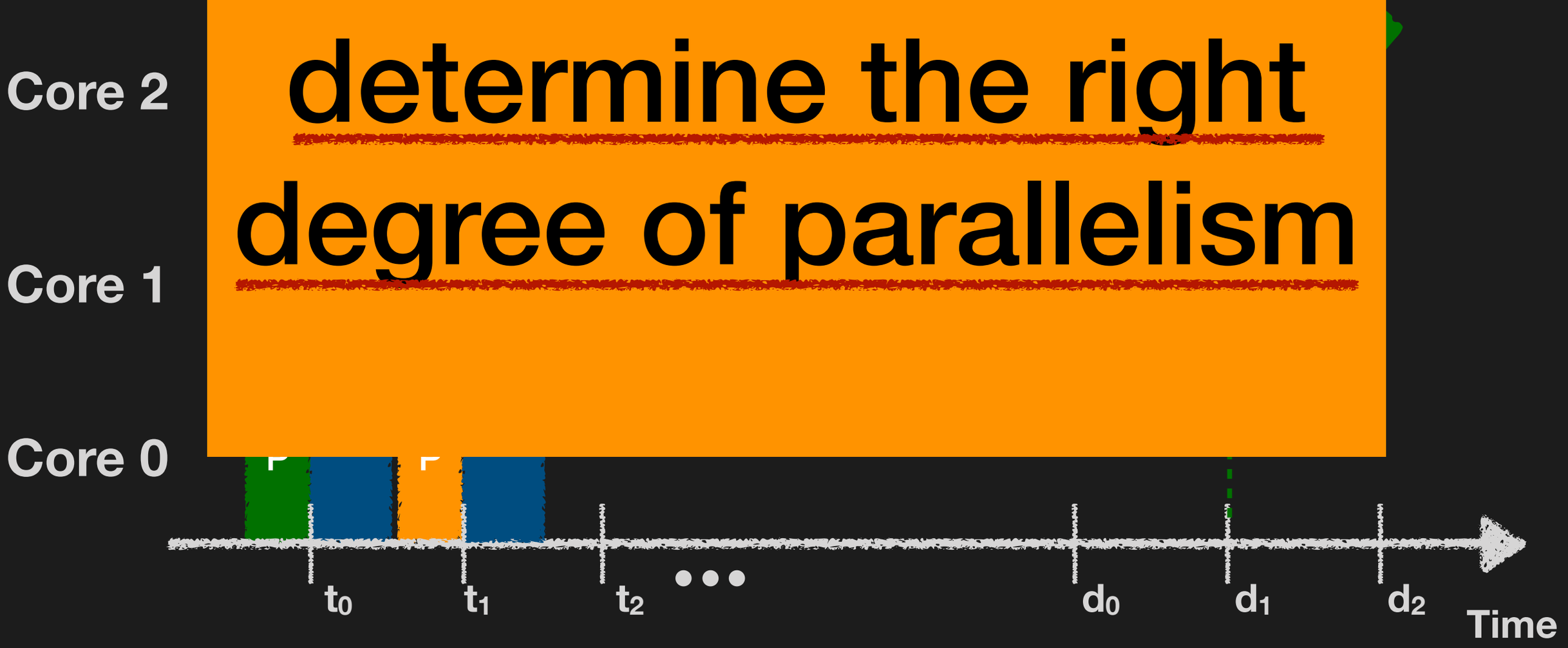


Parallel Real-Time Programming is Hard

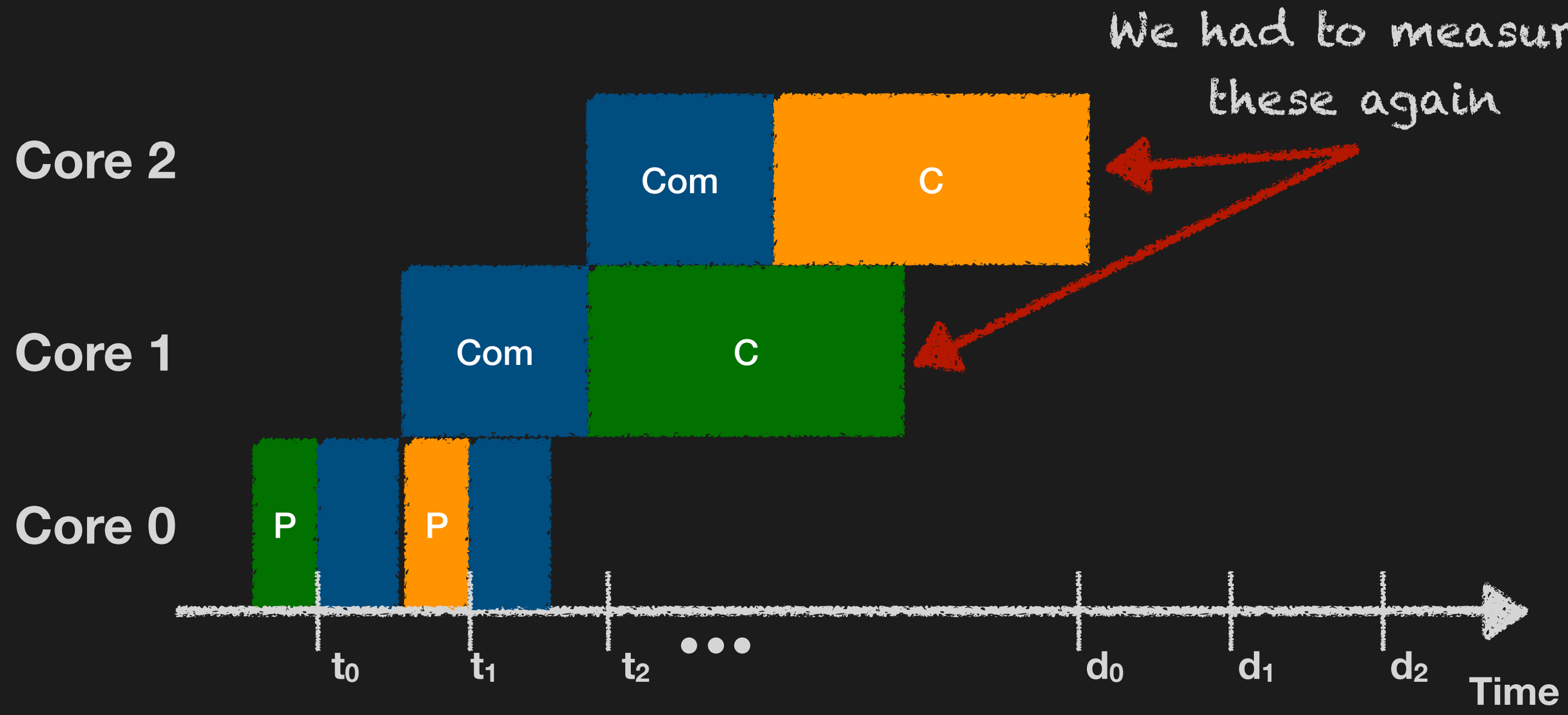


Parallel Real-Time

We have to determine the right degree of parallelism

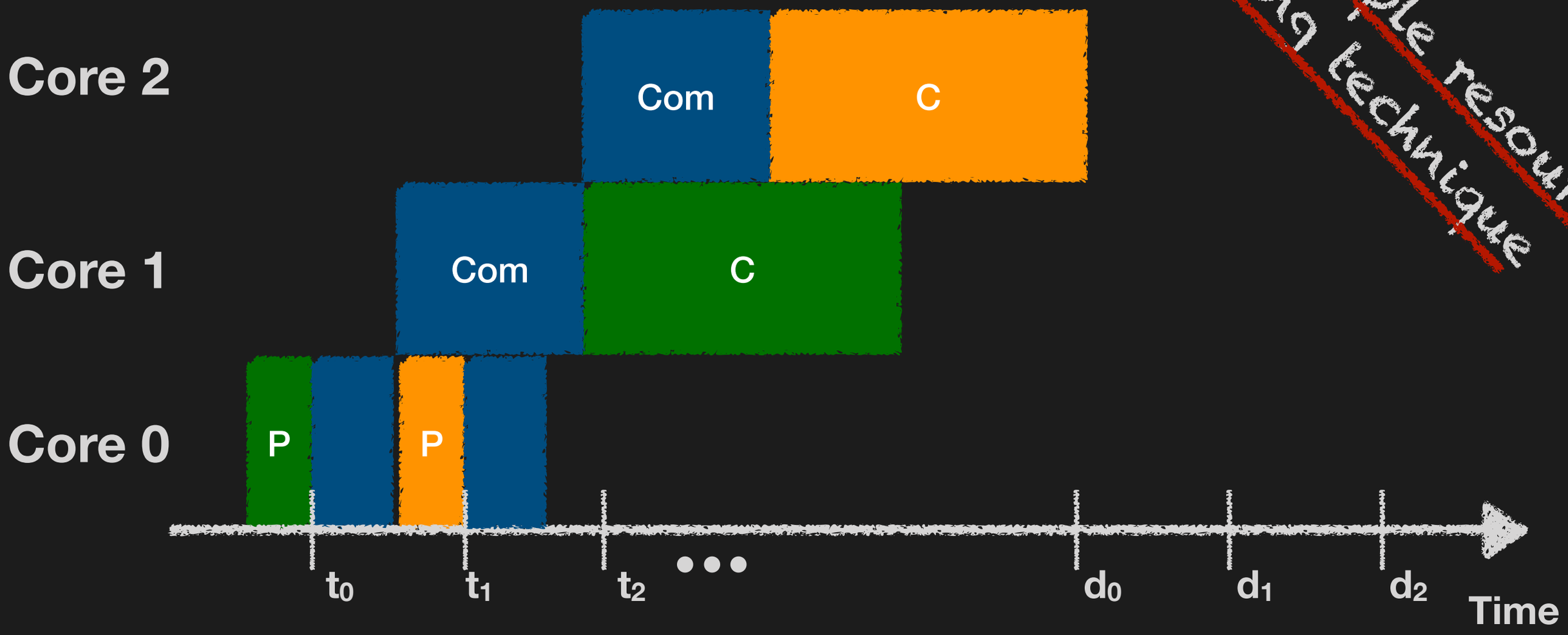


Parallel Real-Time Programming is Hard

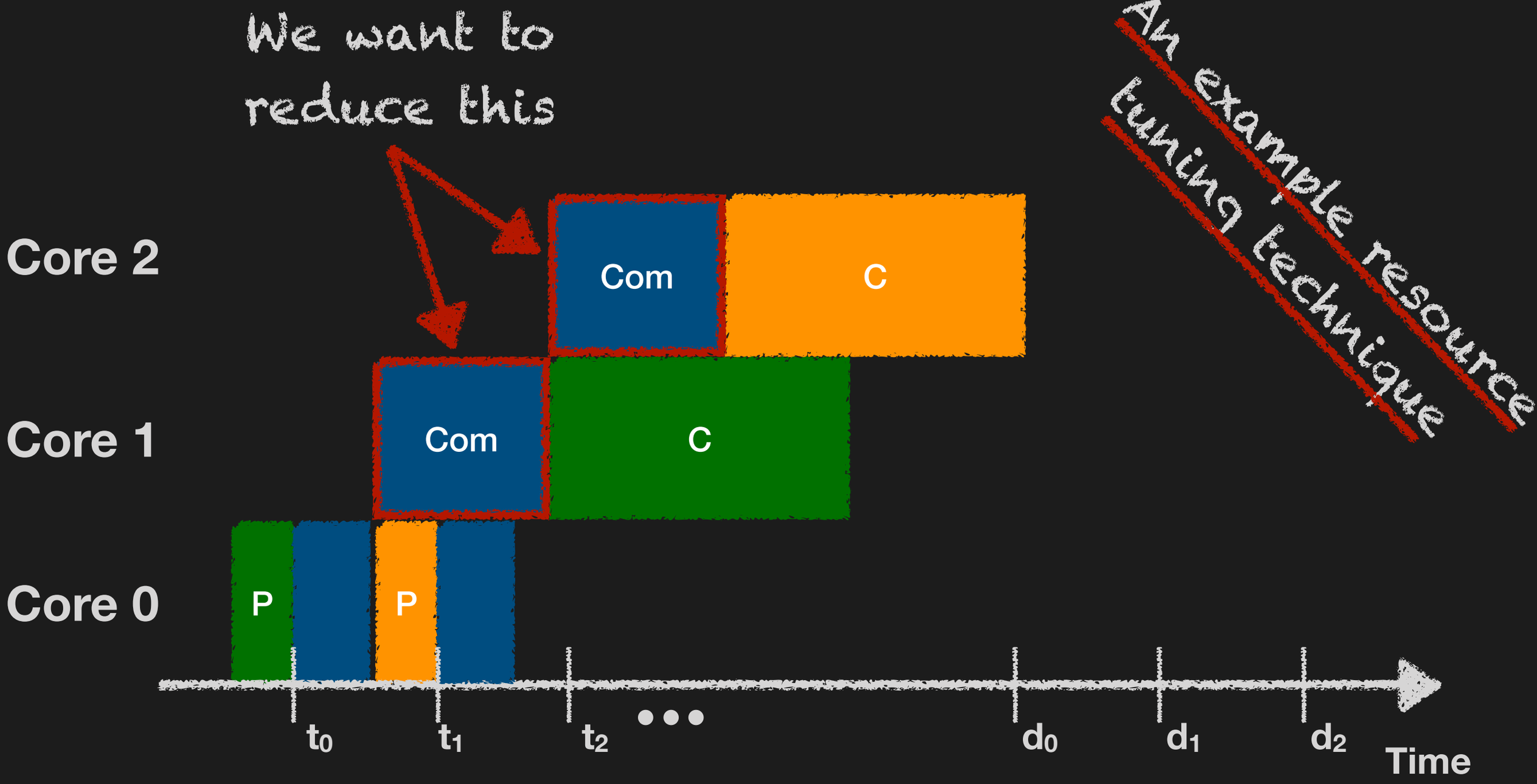


Parallel Real-Time Programming is Hard

An example resource tuning technique



Parallel Real-Time Programming is Hard

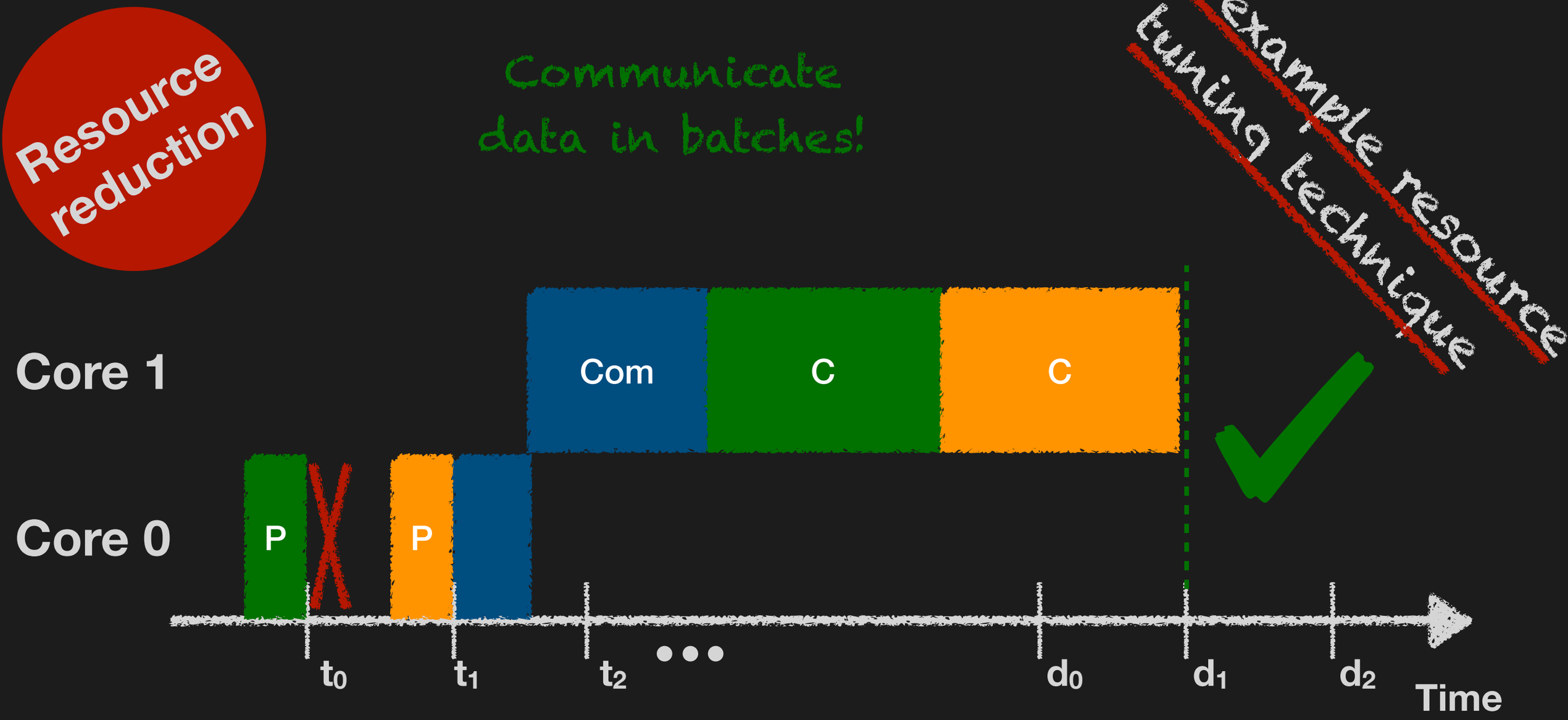


Parallel Real-Time Programming is Hard

Resource reduction

Communicate data in batches!

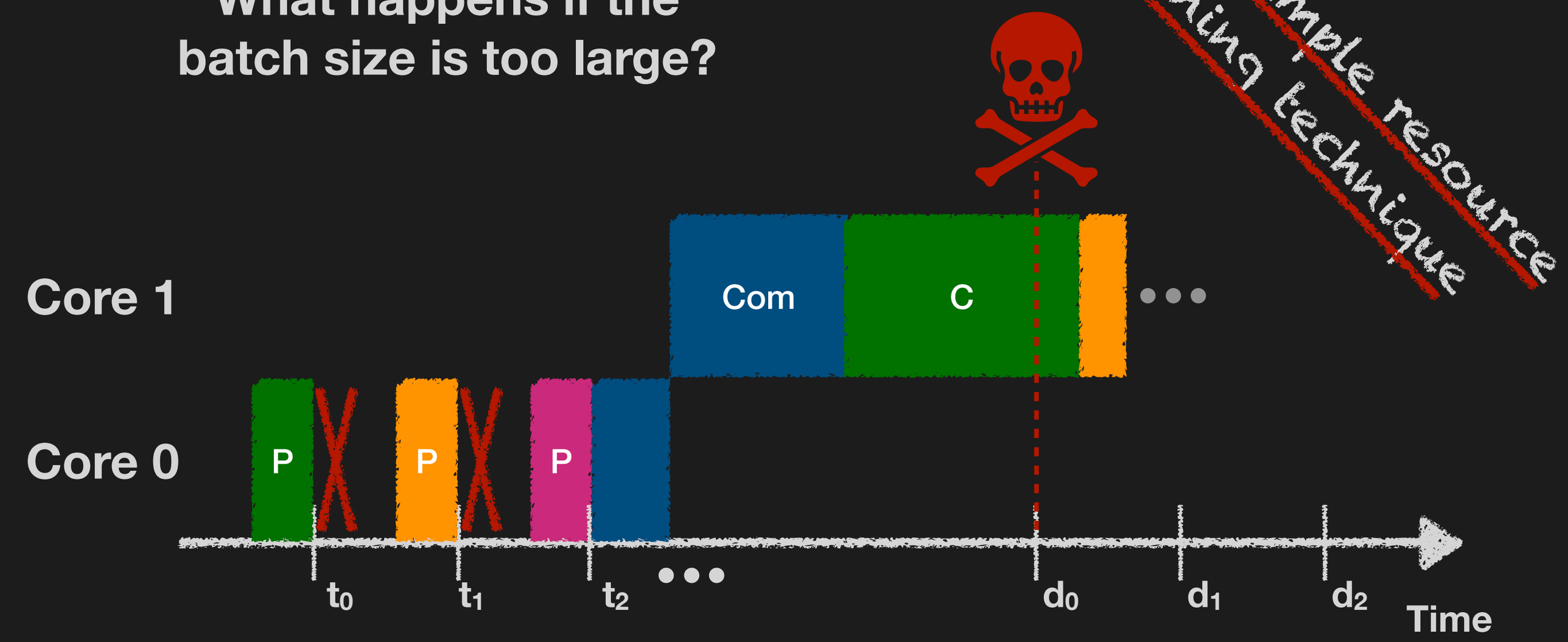
An example resource tuning technique



Parallel Real-Time Programming is Hard

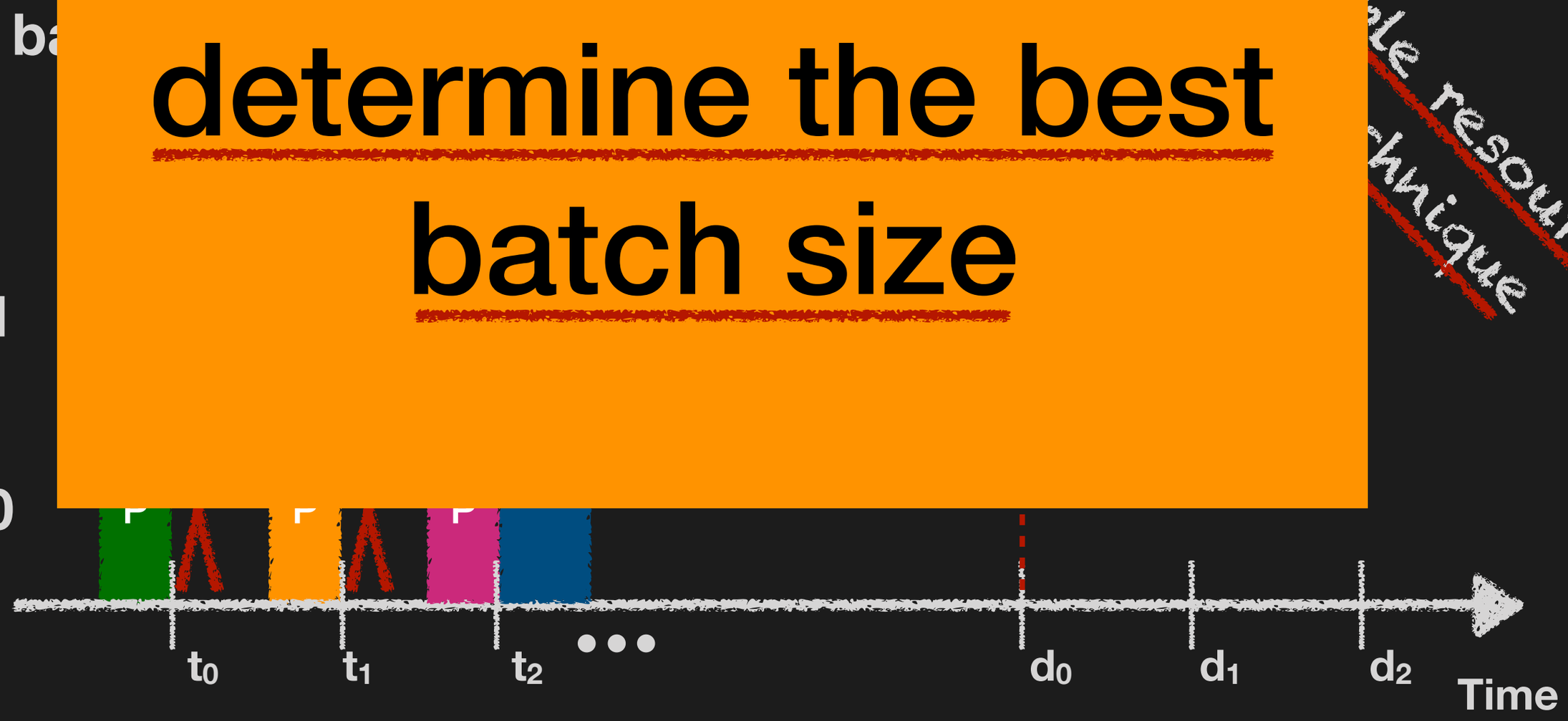
What happens if the batch size is too large?

An example resource tuning technique



Parallel Real-Time

We have to determine the best batch size



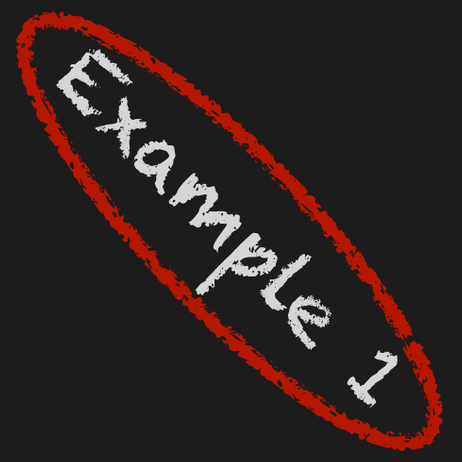
What we want

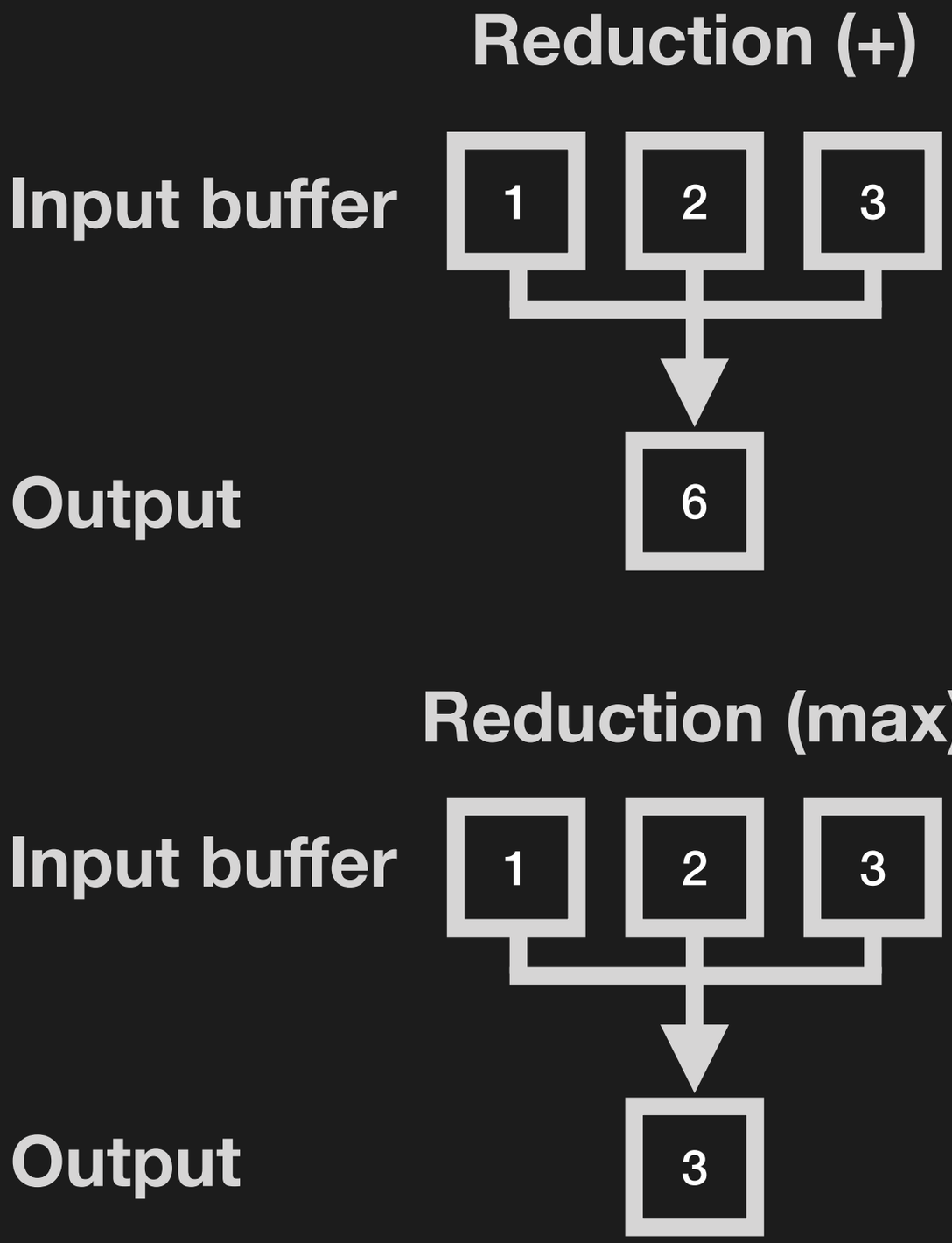
- **Automatically determine the degree of parallelism**
- **Automatically choose tuning parameters such as the batch size**
- **Hide complex parallel code from programmers**

Our solution from ~~another~~ ~~planet~~  general purpose parallel programming:

Skeletons

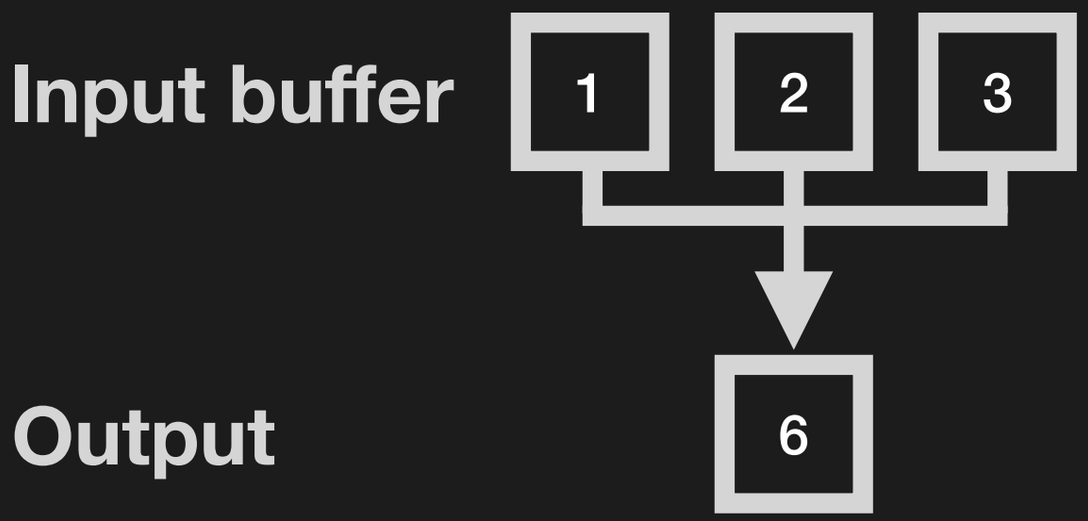
What are Skeletons?



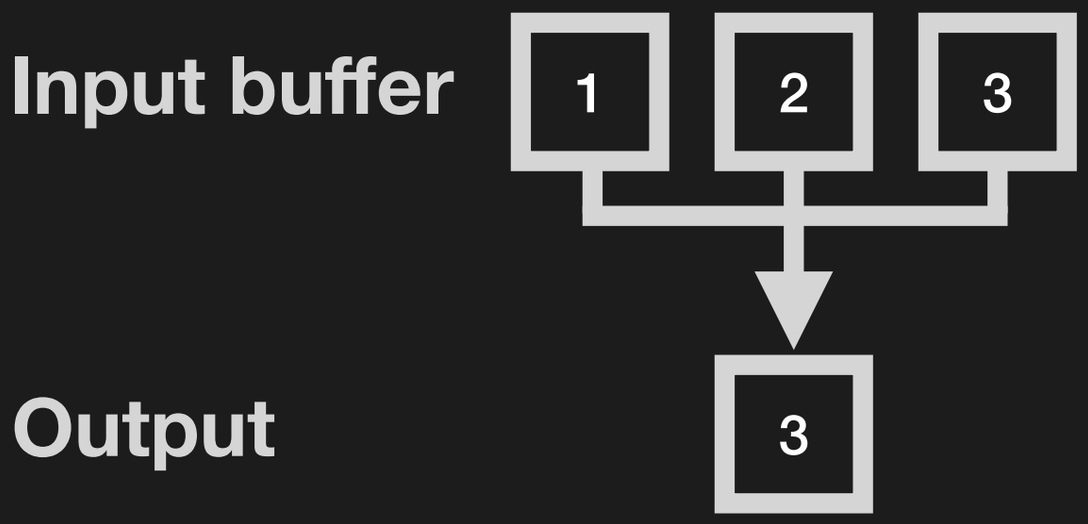


What are Skeletons?

Reduction (+)



Reduction (max)



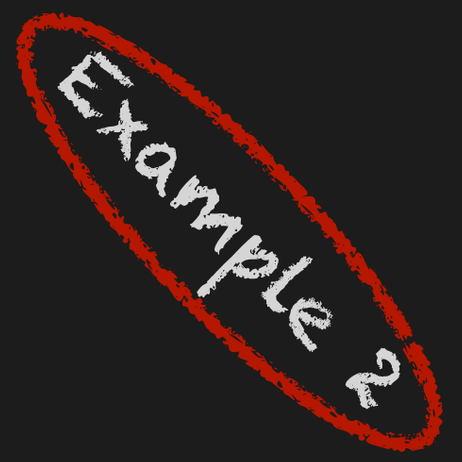

```
…
pthread_t t[THREAD_CNT];
for (…) pthread_create(…);
for (…) t[i].join();
…

…
int idx = …
int offset = …
for (…) {
    partial_result +=
      input_buffer[k];
}
…

…
for (…) result +=
 partial_result[j];
…
```

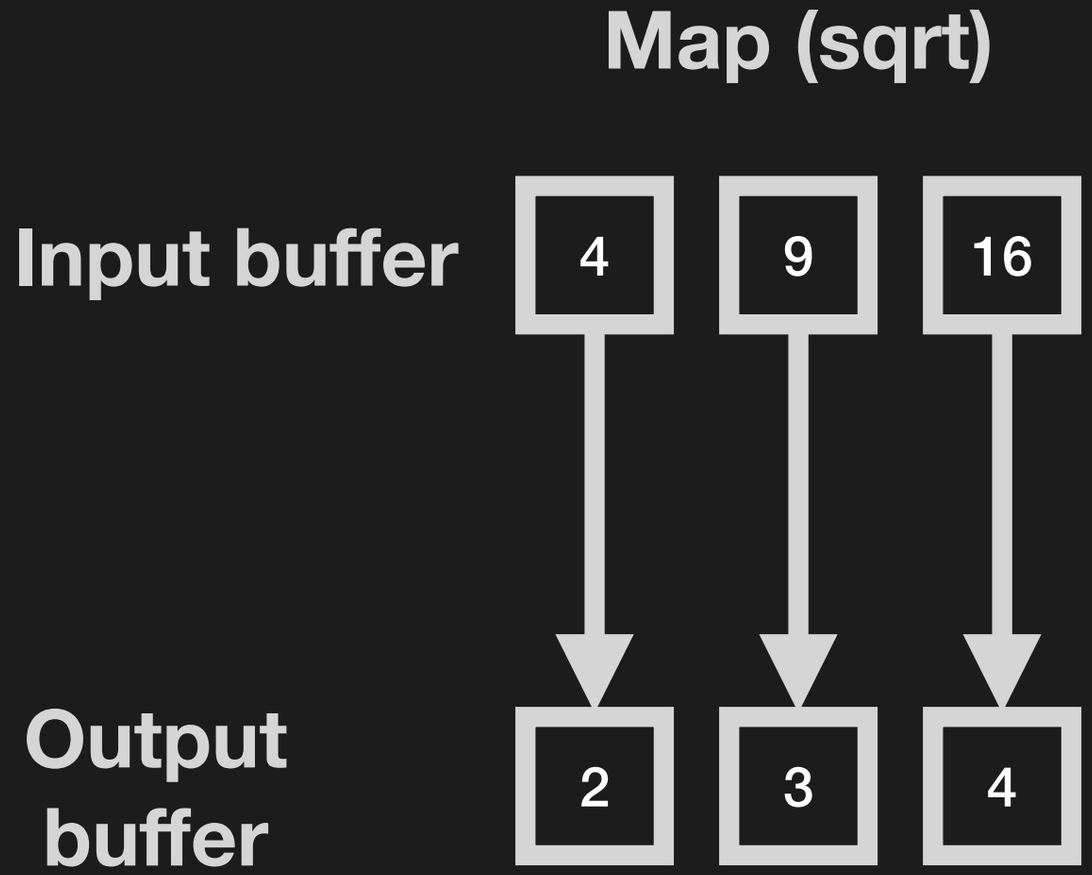
What are Skeletons?

```
reduction_skeleton(
input_buffer,
result,
);
```

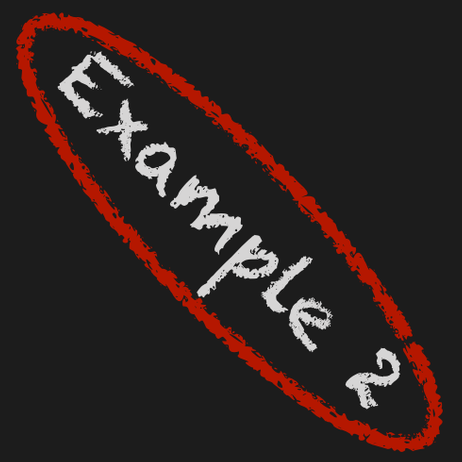
```
…
pthread_t t[THREAD_CNT];
for (…) pthread_create(…);
for (…) t[i].join();
…

…
int idx = …
int offset = …
for (…) {
    partial_result +=
      input_buffer[k];
}
…

…
for (…) result +=
 partial_result[j];
…
```

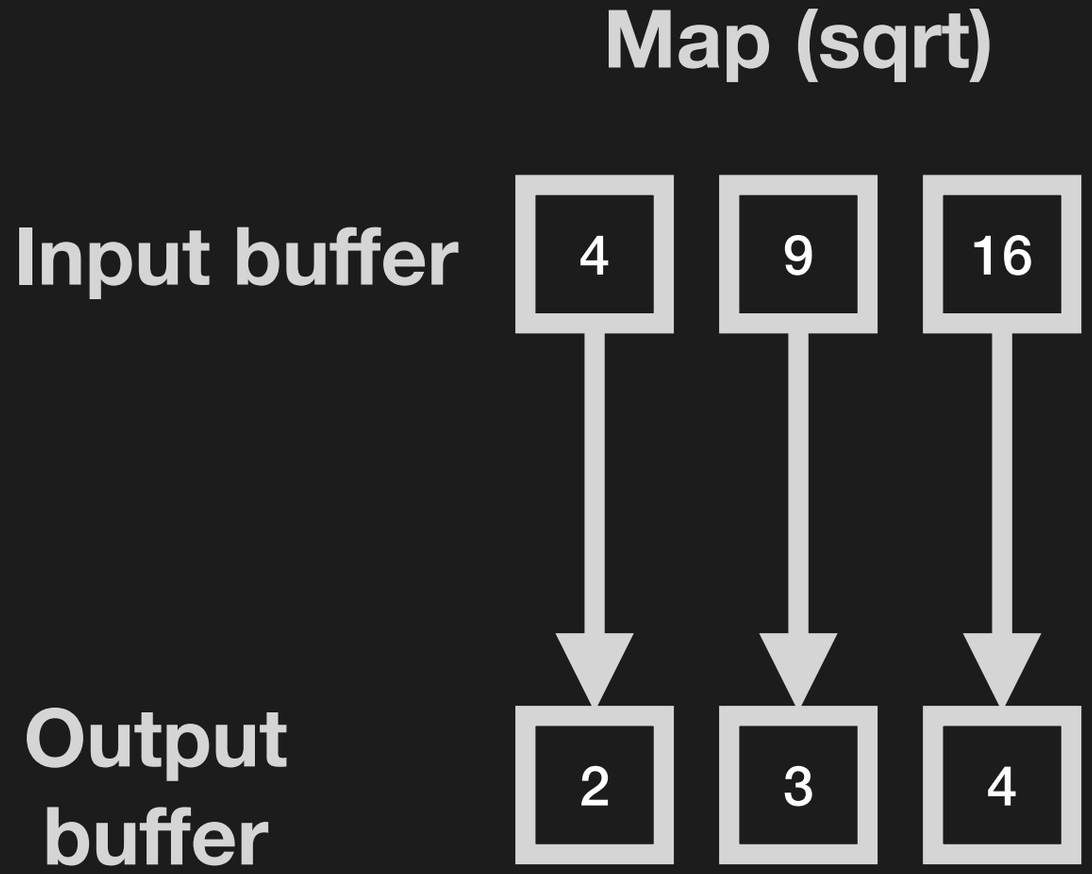
What are Skeletons?





What are Skeletons?

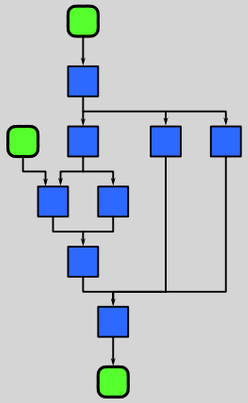




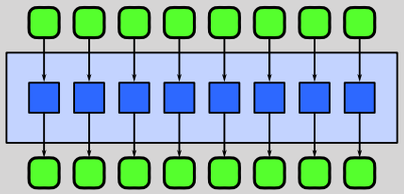

```
map_skeleton(
    input_buffer,
output_buffer,
sqrt);
```

What are Skeletons?

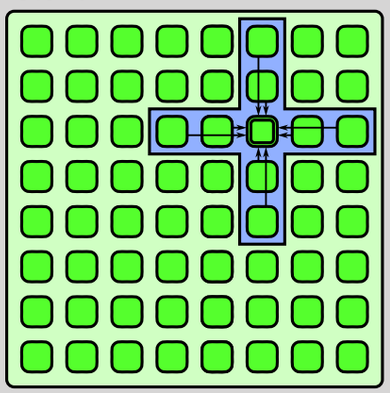
Superscalar sequence



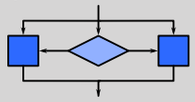
Map



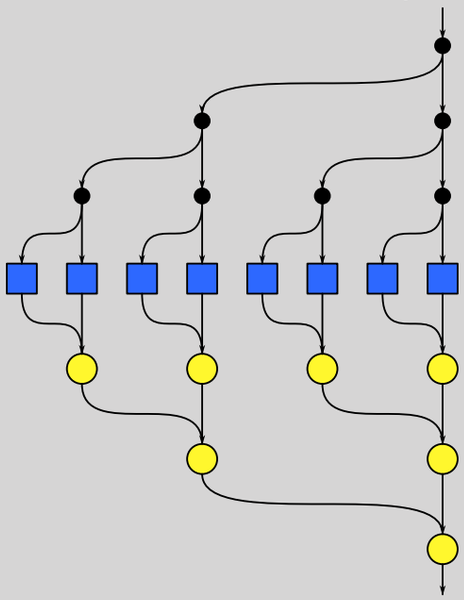
Stencil



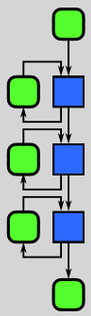
Speculative selection



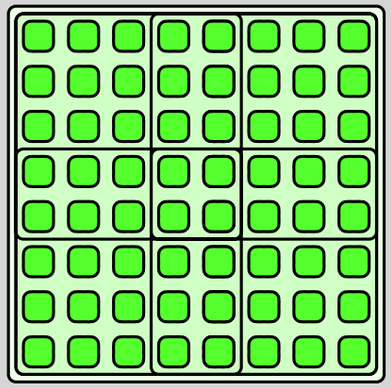
Fork–join



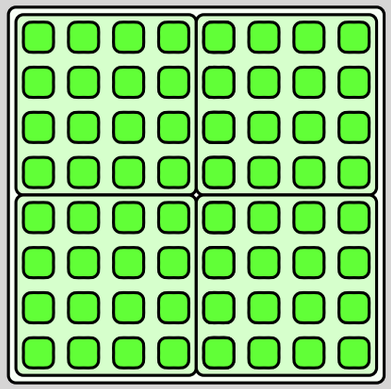
Pipeline



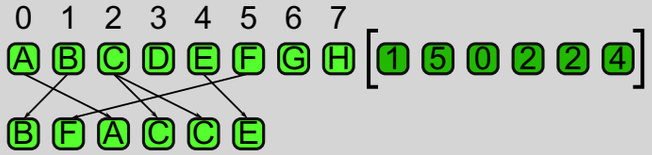
Geometric decomposition



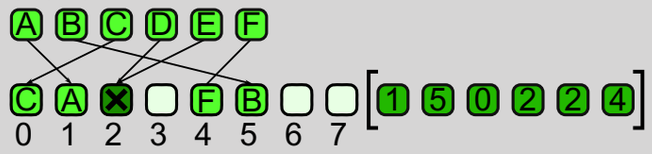
Partition



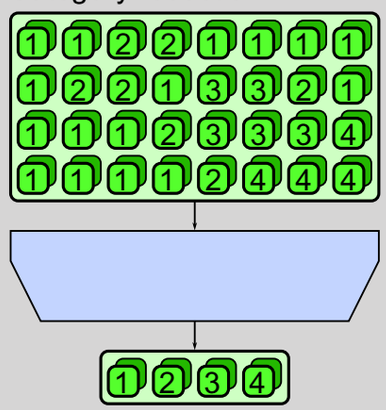
Gather



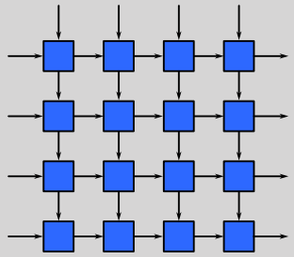
Scatter



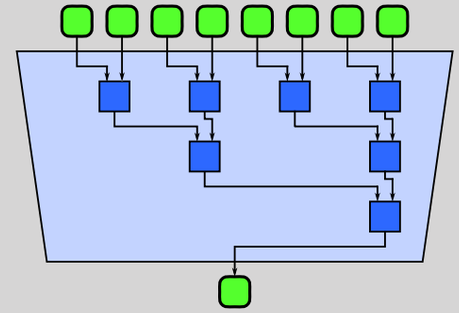
Category reduction



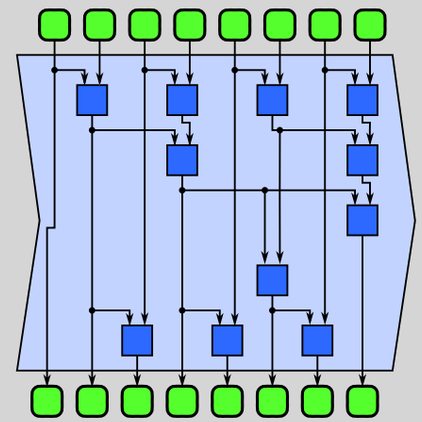
Recurrence



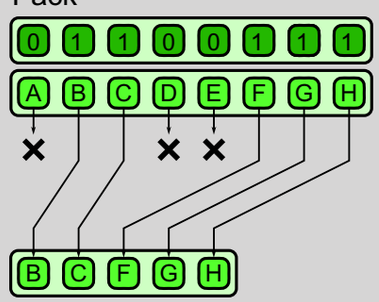
Reduction



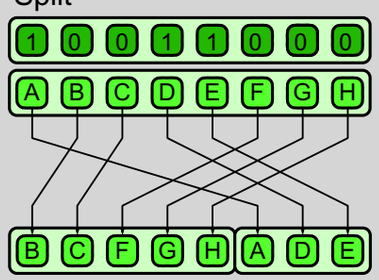
Scan



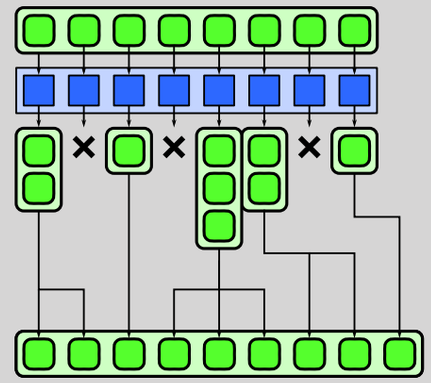
Pack



Split

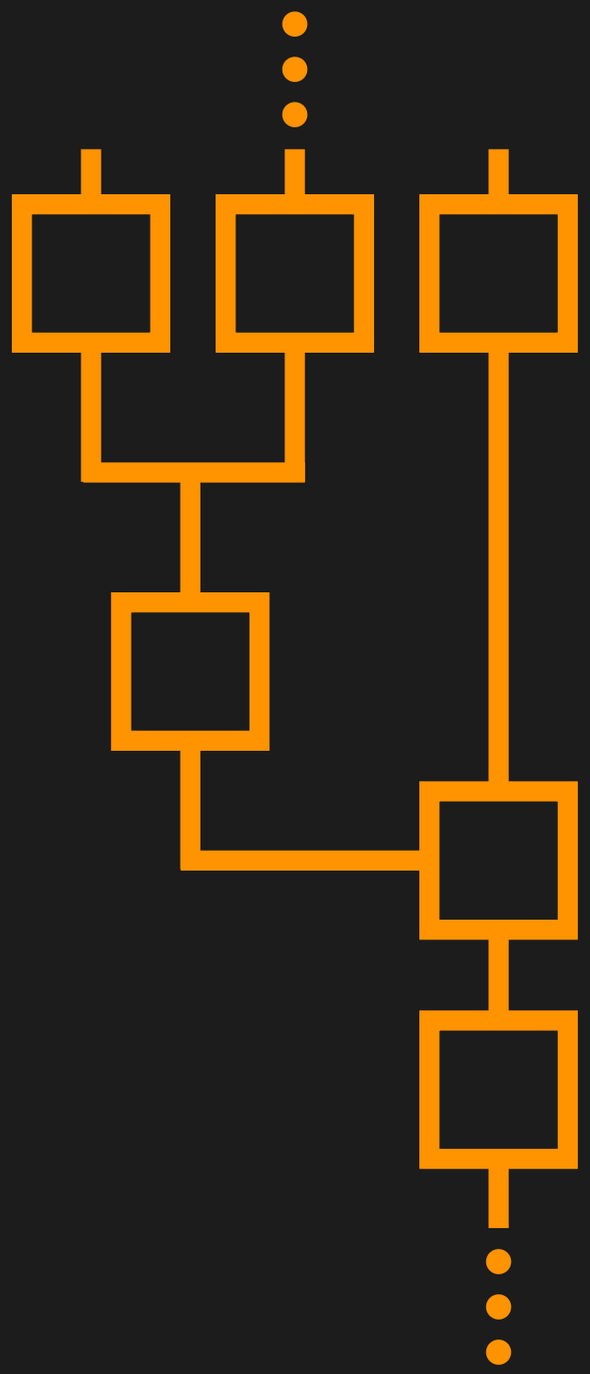


Expand

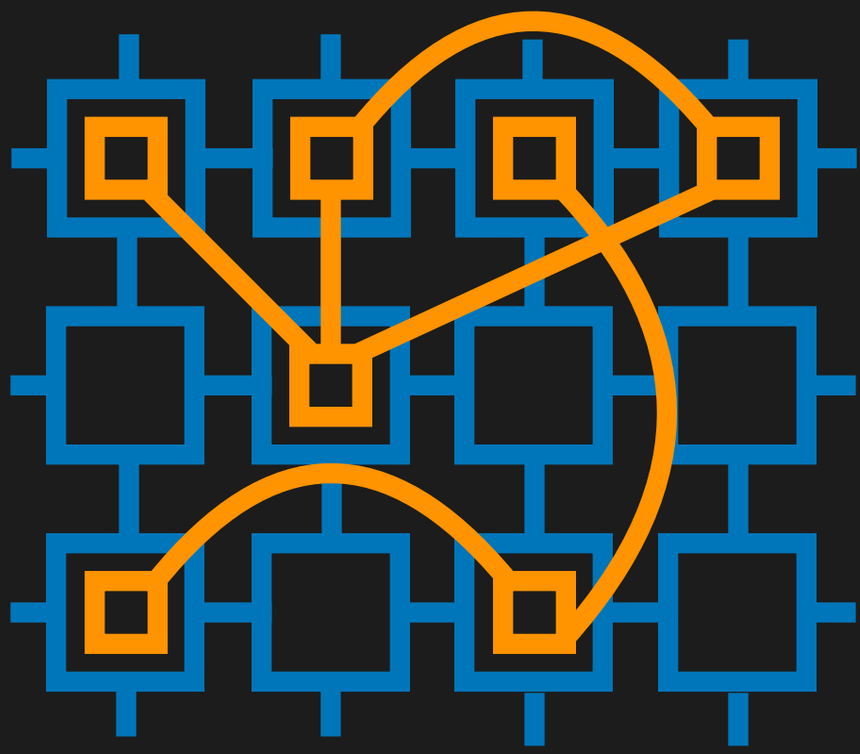


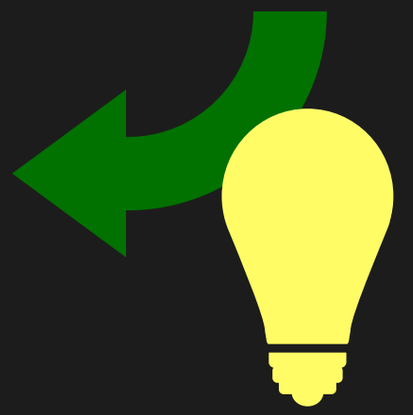
from Structured Parallel Programming: Patterns for Efficient Computation. McCool et al., 2012

Skeleton Tuning



```
…
scan_skeleton(…);
map_skeleton(…);
reduce_skeleton(…);
…
```

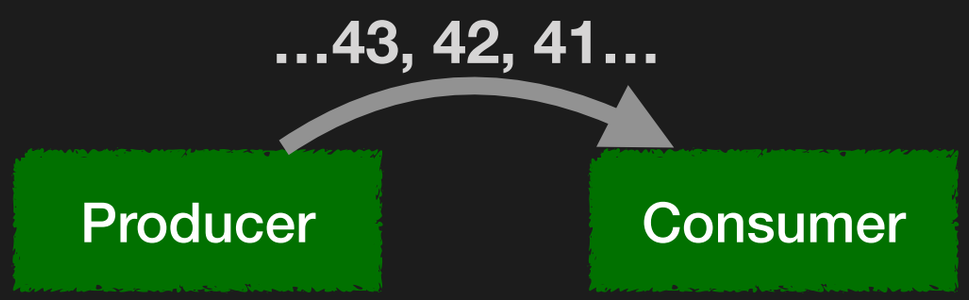




A case study with the producer-consumer skeleton

(We call this skeleton 'job farm' in the paper.)

Producer-Consumer Case Study




```
producer_consumer_skeleton(
 producer_func, consumer_func,
 consumer_wcet);
```

Producer-Consumer Case Study

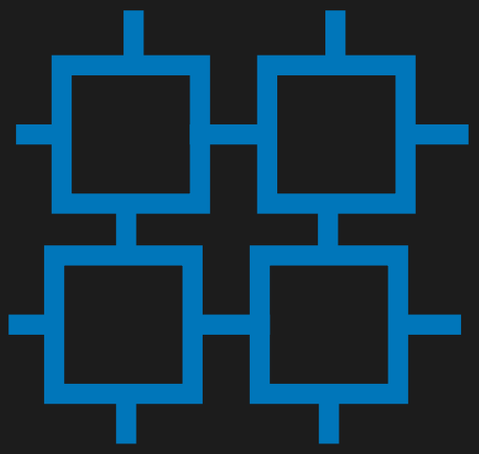
`producer_consumer_skeleton(…);`

Ease of programming

```
…
int idx = …
int offset = …
for (…) {
    partial_result +=
      input_buffer[k];
}
…

…
pthread_t t[THREAD_CNT];
for (…) pthread_create(…);
for (…) t[i].join();
…
```

Batch size

Degree of Parallelism



Producer-Consumer Case Study

```
producer_consumer_skeleton(…);
```

No implementation specified

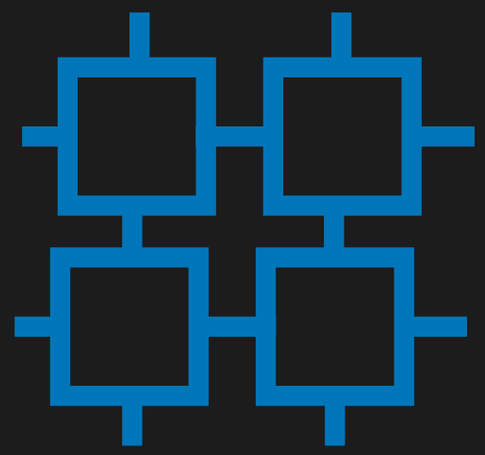
Set without programmer intervention

Ease of programming

```
…
int idx = …
int offset = …
for (…) {
    partial_result +=
      input_buffer[k];
}
…

…
pthread_t t[THREAD_CNT];
for (…) pthread_create(…);
for (…) t[i].join();
…
```

Batch size

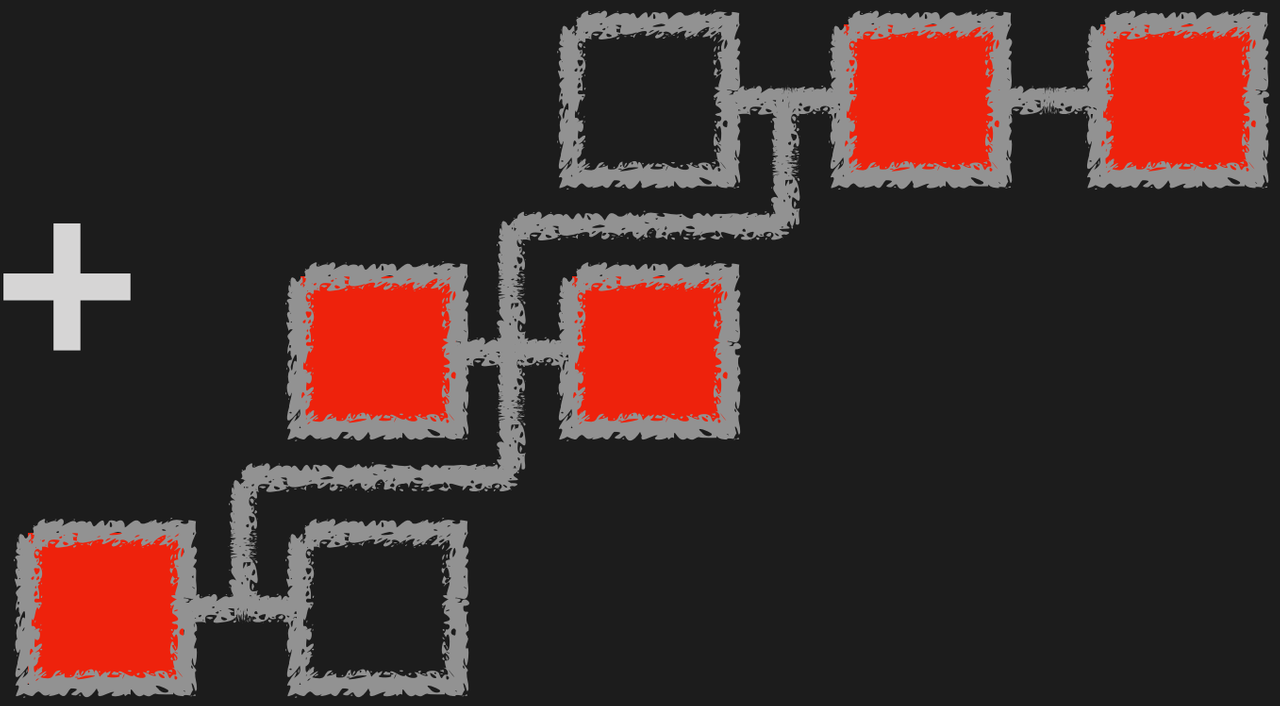
Degree of Parallelism



How do we do it?

of skeleton-internal code

WCETs +



consumer code provided by the programmer

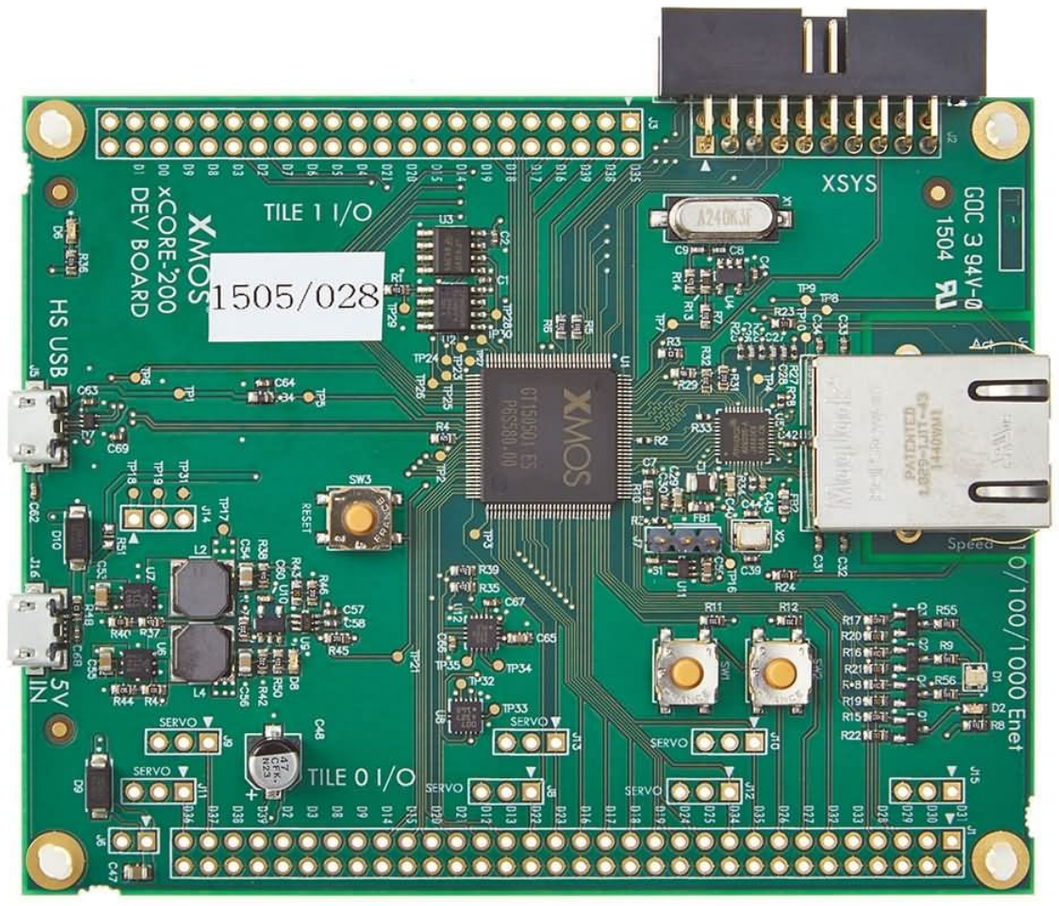
Skeleton specific critical path

How do we do it?

Analytical models for the { best DoP ; best batch size

Results

XMOS xCore-200



Source: https://www.xmos.com/wp-content/uploads/2019/04/explorerkit-200.jpg

Real time:

- **Hard real time**
- **Static analysis for WCETs**
- **No data caches**
- **No OS**

Parallelism:

- **Fine grained multithreading**
- **Eight logical cores**
- **Cores share address space**

We find the best

We use real hardware

degree of parallelism in

89%

batch size in

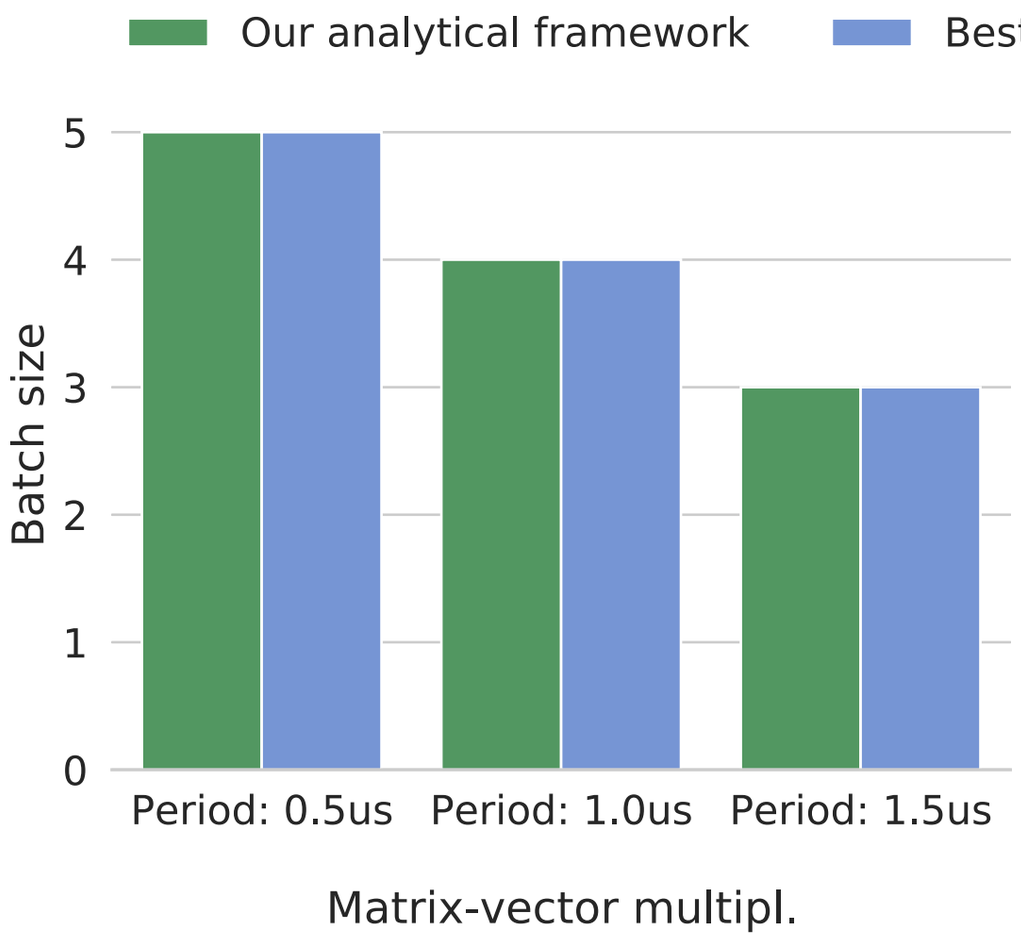
72%

of all cases.

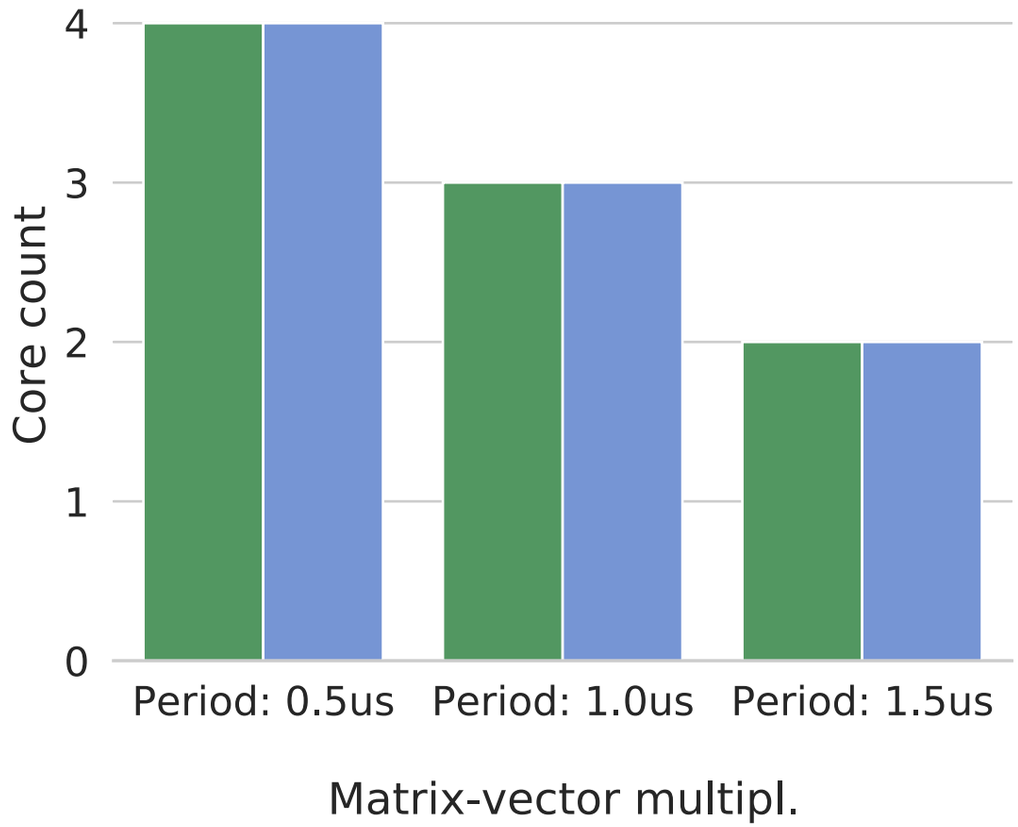
We never cause deadline misses

Excerpt: Matrix Multipl.

Batch size choices

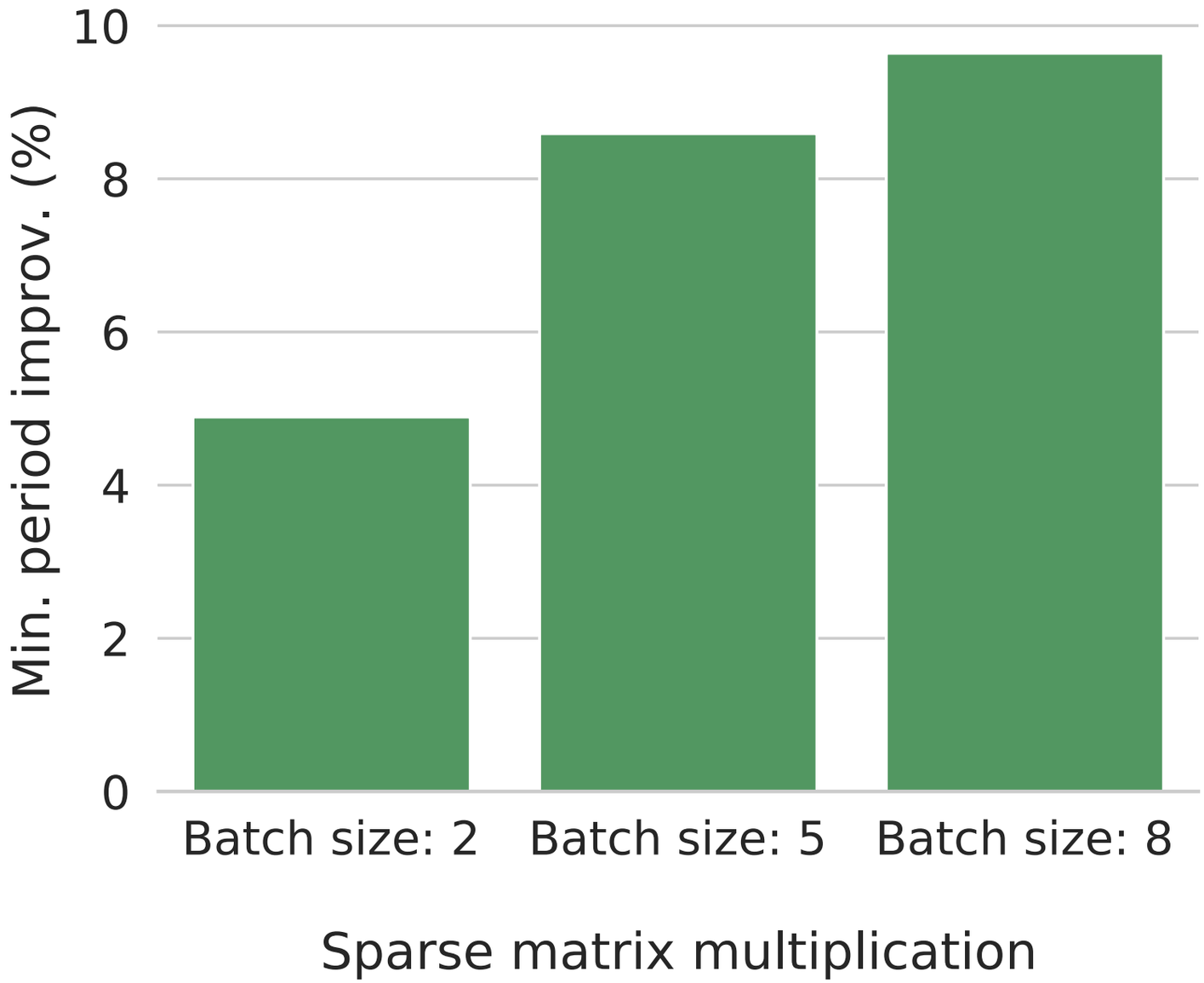


DoP choices



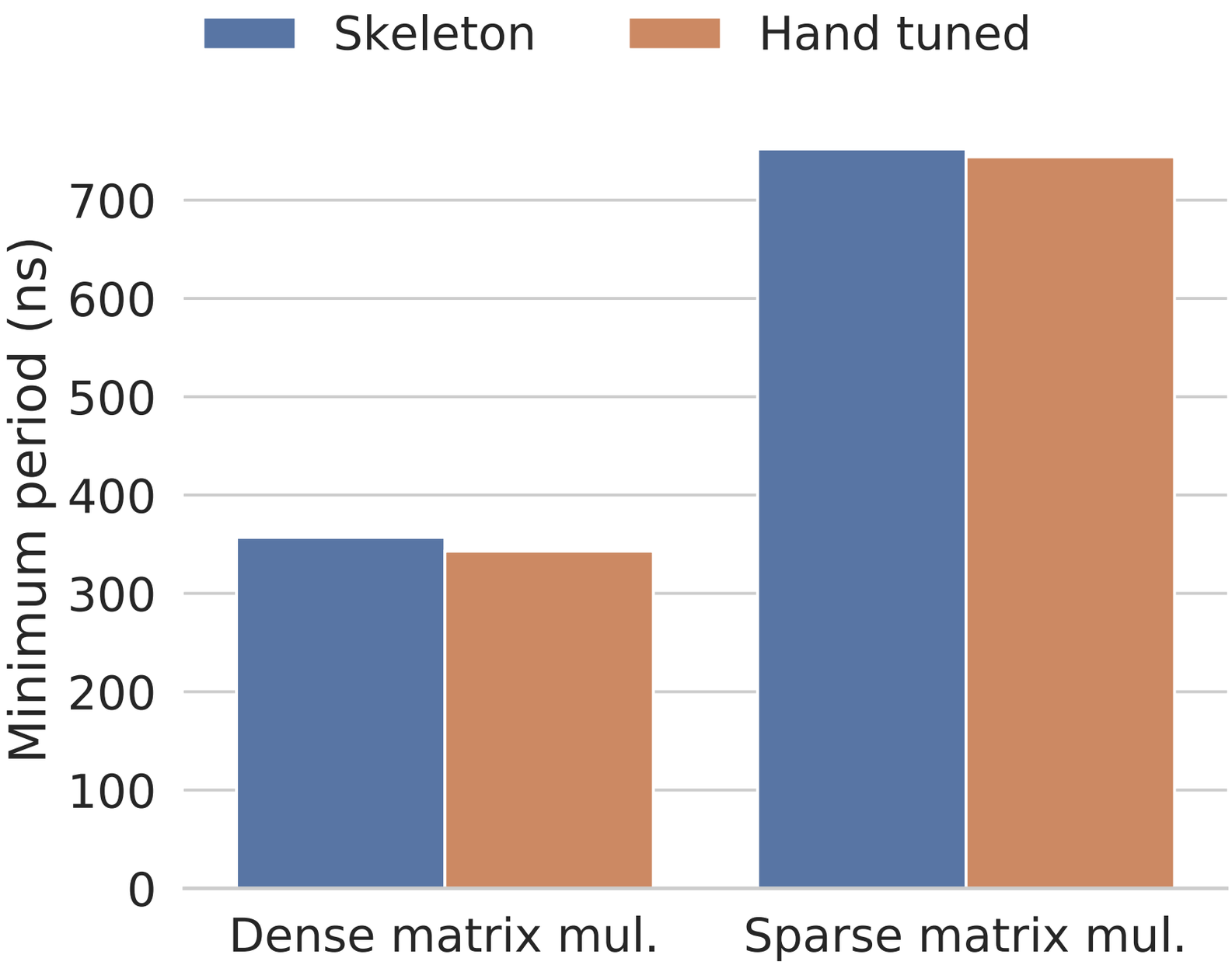
More benchmarks in the paper

Excerpt: Batching



Max. improvement with our benchmarks: 22%

Excerpt: Overheads



Avg. overheads: 3.48%

What we want

- Automatically determine the degree of parallelism ✓
- Automatically choose tuning parameters such as the batch size ✓
- Hide complex parallel code from programmers ✓

Future work

More real-time skeletons

Being able to compose them

Paul Metzger **University of Edinburgh**

Enforcing Deadlines for Skeleton-based Parallel Programming

Code is available online

rtas2020.paulmetzger.info