Resource Management with Makeflow & Work Queue

Ben Tovar
University of Notre Dame

btovar@nd.edu

CCTools

UNIVERSITY OF NOTRE DAME

Resources Makeflow and WQ care about

cores

memory

disk

Resources contract

```
Worker has
available:

i cores
j MB of memory
k MB of disk
```

```
Task needs:


m cores
n MB of memory
o MB of disk
```

Task runs only if it fits in the currently available worker resources.

Resources contract example

```
Worker has
available:

8 cores
512 MB of memory
512 MB of disk
```

```
Task a:

4 cores
100 MB of memory
100 MB of disk
```

```
Task b:

3 cores
100 MB of memory
100 MB of disk
```

Tasks a and b may run in worker at the same time.
(Work could still run another 1 core task.)

Beware!
Tasks use all worker on missing declarations

```
Worker has
available:

8 cores
512 MB of memory
500 TB of disk
```

```
Task a:

4 cores
100 MB of memory
```

```
Task b:

3 cores
100 MB of memory
```

Tasks a and b may NOT run in worker at the same time.
(disk resource is not specified.)

Resource Management Levels

Do nothing (default):

One task per worker, task occupies the whole worker.

Honor contract:

Both worker and task declare resources (cores, memory, disk).
Worker runs as many concurrent tasks as they fit.
Tasks **may** use more resources than declared.

Monitoring and Enforcement:

Tasks fail (permanently) if they go above the resources declared.

Automatic resource labeling:

Tasks are retried with resources that maximize throughput, or minimize waste.

Declaring resources: worker

By default, a worker declares:

1 core

All physical memory (RAM)

All free disk

Declaring resources: worker

--cores=# of cores
--memory=MB of RAM
--disk=MB of disk

```
% work_queue_worker ... --cores 4 ...
% sge_submit_workers ... --memory 1024 ...
% work_queue_factory ... --cores all --disk 20000
```

Declaring resources: worker

```
% export CORES=8
% export MEMORY=1024
% export DISK=20000
%
% work_queue_worker ...
% sge_submit_workers ...
% work_queue_factory ...
```

Declaring resources: tasks

Tasks are grouped into **categories**.

All tasks in a category have identical resource requirements.

Unless specified otherwise, all tasks belong to the "**default**" category.

Categories

my_category

```
Task a:

4 cores
100 MB of memory
100 MB of disk
```

```
Task b:

4 cores
100 MB of memory
100 MB of disk
```

my_other_category

```
Task c:

1 cores
200 MB of memory
512 MB of disk
```

Declaring resources (Makeflow)

```
# Makeflow file
# Resources for "default" category
.MAKEFLOW CORES  4
.MAKEFLOW MEMORY 1024
.MAKEFLOW DISK   1024

# all rules run with 4 cores, 1024 MB RAM, etc.
output_a: input_a
   cmd < input_a > output_a

output_b: input_b
   cmd < input_b > output_b
```

```
# Makeflow file

.MAKEFLOW CATEGORY MY_FIRST_CATEGORY
.MAKEFLOW CORES  1
.MAKEFLOW MEMORY 1024
.MAKEFLOW DISK   1024

.MAKEFLOW CATEGORY MY_SECOND_CATEGORY
.MAKEFLOW CORES  2
.MAKEFLOW MEMORY 2048
.MAKEFLOW DISK   4096

.MAKEFLOW CATEGORY MY_FIRST_CATEGORY
output_a: input_a
    cmd < input_a > output_a

output_b: input_b
    cmd < input_b > output_b

.MAKEFLOW CATEGORY MY_SECOND_CATEGORY
output_c: input_c
    cmd < input_c > output_c
```

Categories group tasks with the identical resource requirements.

Resource declarations are assigned to the latest CATEGORY=...

These tasks belong to MY_FIRST_CATEGORY

This task belongs to MY_SECOND_CATEGORY

Example

```
% makeflow -Twq Makeflow

% # launch a worker
% work_queue_worker HOST PORT --cores 1

% # launch a bigger worker
% work_queue_worker HOST PORT --cores 2
```

work_queue_status -A HOST PORT

information about waiting tasks and resources

```
CATEGORY RUNNING WAITING FIT-WORKERS  MAX-CORES MAX-MEM MAX-DISK
my-cat-a       2      20           2          1   ~1024    ~2000
```

Number of workers able to eventually run a task in the category

~ No hard limit set, but all the tasks have run at most with these resource usage.

Declaring resources (Work Queue)

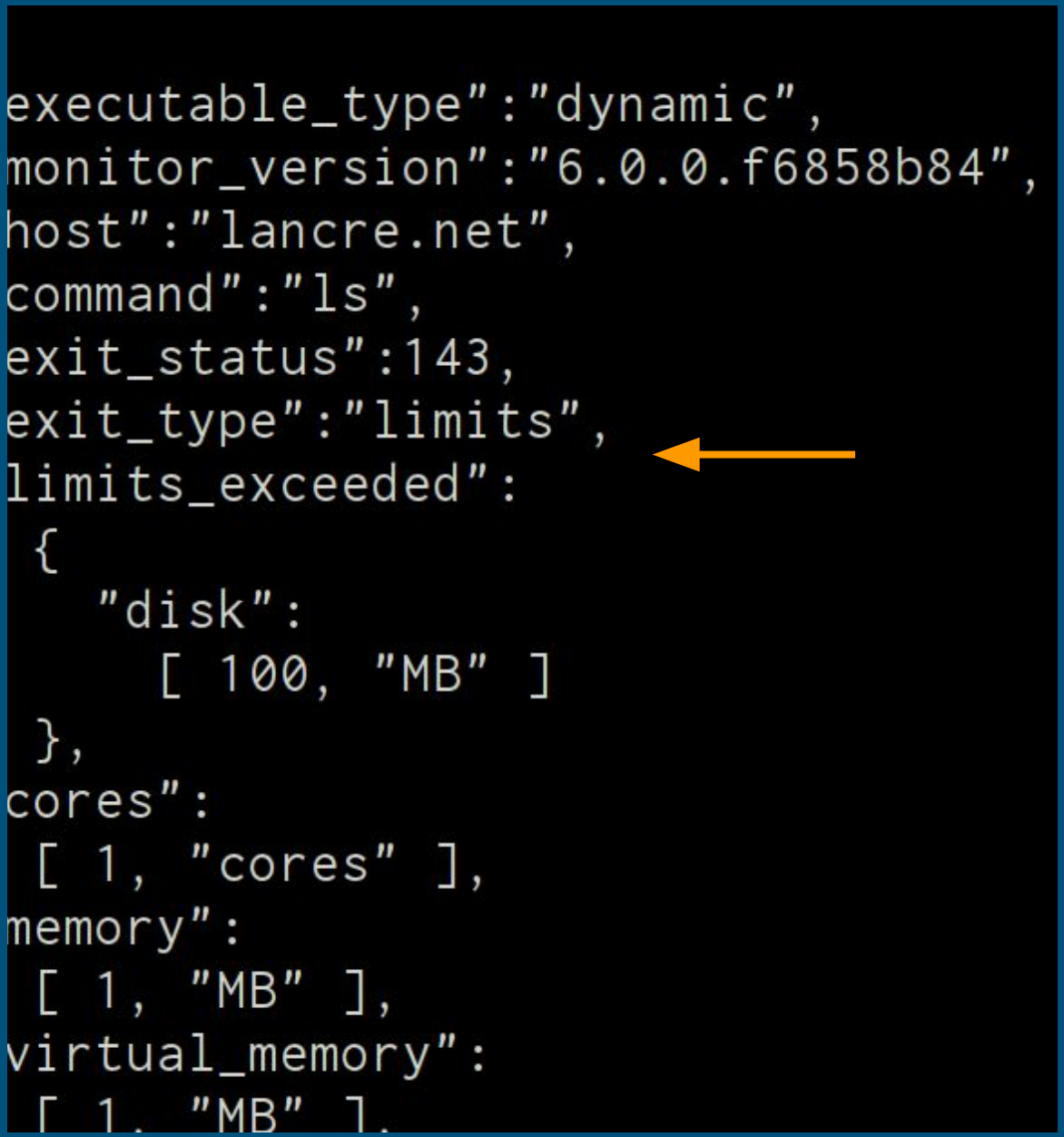
```
q = WorkQueue(port)

q.specify_category_max_resources('my_category', {
   'cores' : 1,
   'memory': 1024,
   'disk'  : 1014
})

t = Task(cmd)
t.specify_category('my_category')
```

Resource Measure and Enforcement

```
% makeflow -Twq --monitor=my_dir Makeflow

% # one resource summary per rule:
% cat mydir/resource-rule-2.summary
```

```
{ "executable_type":"dynamic",
  "host":"lancre.net",
  "command":"ls",
  "exit_status":0,
  "exit_type":"normal",
  "wall_time":
    [ 0.005001, "s" ],
  "cpu_time":
    [ 0, "s" ],
  "cores":
    [ 1, "cores" ],
  "memory":
    [ 3, "MB" ],
  "virtual_memory":
    [ 17, "MB" ],
  "swap_memory":
    [ 0, "MB" ],
  "bytes_read":
```


Task finished in the allotted resources.



Task exhausted its resources.

Monitor and Enforcement with Work Queue

```
q = WorkQueue(port)
q.enable_monitoring('my_summaries_dir')

t = q.wait(timeout)

t.resources_allocated.cores #.memory, .disk,
etc.
t.resources_measured.memory

# resources exhausted, if any.
if t.limits_exceeded:
  t.limits_exceeded.wall_time
```

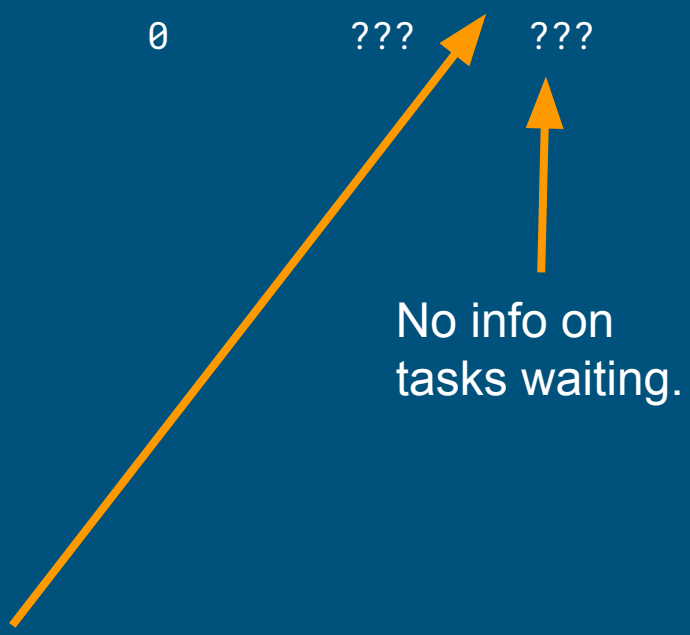
Other resources measured

```
# start:                  microseconds at the start of execution
# end:                    microseconds at the end of execution
# wall_time:              microseconds spent during execution
# cpu_time:               user + system time of the execution
# cores:                  number of cores. Sliding window of cpu_time/wall_time
# max_concurrent_processes: the maximum number of processes running concurrently
# total_processes:        count of all of the processes created
# virtual_memory:         maximum virtual memory across all processes
# memory:                 maximum resident size across all processes
# swap_memory:            maximum swap usage across all processes
# bytes_read:             number of bytes read from disk
# bytes_written:          number of bytes written to disk
# bytes_received:         number of bytes read from the network
# bytes_send:             number of bytes written to the network
# bandwidth:              maximum network bits/s (average over one minute)
# workdir_num_files:      maximum number of files and directories
# workdir_footprint:      size in MB of all working directories in the tree
```

work_queue_status -A HOST PORT

information about waiting tasks and resources

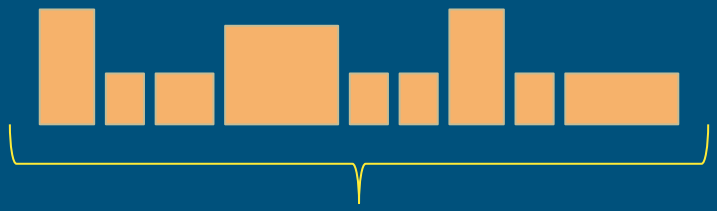
```
CATEGORY RUNNING WAITING FIT-WORKERS  MAX-CORES MAX-MEM MAX-DISK
my-cat-a       2      20           2          1   ~1024    ~2000
my-cat-b       0      15           0          1   >3000    ~1000
my-cat-c       0       0           0        ???     ???      ???
```



No info on tasks waiting.

> At least one task that is now waiting, failed exhausting these much of the resource.

Tasks with Unknown Resource Requirements

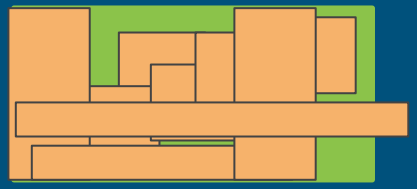


Tasks which size
(e.g., cores, memory, and disk)
is not known until runtime.

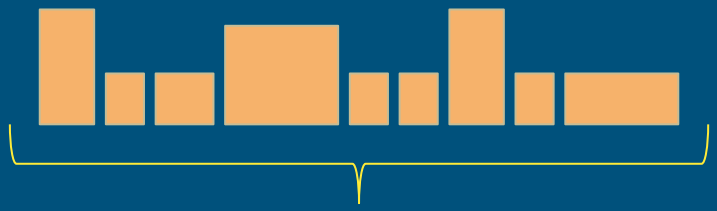
workers

One task per worker:
Wasted resources, reduced throughput.

Many tasks per worker:
Resource contention/exhaustion, reduce throughput



Tasks with Unknown Resource Requirements



Tasks which size
(e.g., cores, memory, and disk)
is not known until runtime.

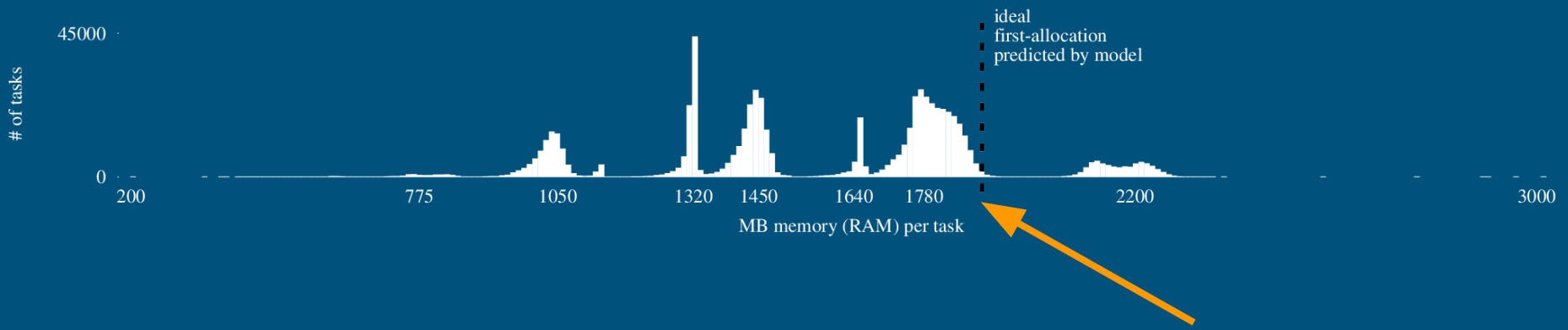
workers

1. Run some tasks using full workers.
2. Collect statistics.
3. Guess task sizes to maximize throughput, or minimize waste.
    a. Run task using guessed size.
    b. If task exhausts guessed size, keep retrying on full (bigger) workers.
4. When statistics become out-of-date, go to 1.

ND CMS example

Real result from a production High-Energy Physics CMS analysis (Lobster NDCMS)

Histogram Peak Memory vs Number of Tasks
O(700K) tasks that ran in O(26K) cores managed by WorkQueue/Condor.



First-allocation that maximizes expected throughput
(increase of %40 w.r.t. no task is retried)

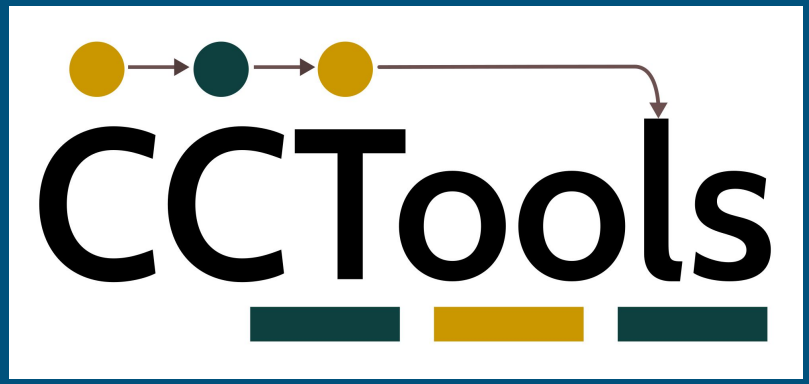
```
# Makeflow file

.MAKEFLOW CATEGORY MY_FIRST_CATEGORY
.MAKEFLOW MODE MAX_THROUGHPUT
.MAKEFLOW CATEGORY MY_SECOND_CATEGORY
.MAKEFLOW MODE MIN_WASTE
.MAKEFLOW CATEGORY MY_OTHER_CATEGORY
.MAKEFLOW MODE FIXED

.MAKEFLOW CATEGORY MY_FIRST_CATEGORY
output_a: input_a
    cmd < input_a > output_a

.MAKEFLOW CATEGORY MY_SECOND_CATEGORY
output_b: input_b
    cmd < input_b > output_b

.MAKEFLOW CATEGORY MY_OTHER_CATEGORY
output_c: input_c
    cmd < input_c > output_c
```

```
% makeflow --monitor=my_dir --retry-count=5
```

Automatic Resource Labels with Work Queue

```
q.enable_monitoring('my_summaries_dir')

q.specify_category_mode('my_cat_a',
WORK_QUEUE_ALLOCATION_MODE_MAX_THROUGHPUT)

q.specify_category_mode('my_cat_b',
WORK_QUEUE_ALLOCATION_MODE_MIN_WASTE)

q.specify_category_mode('my_cat_c',
WORK_QUEUE_ALLOCATION_MODE_FIXED)

# recommended. contains history of allocations
q.specify_transactions_log('transactions.log')

# setting some maximum  # retries is recommended
t.specify_max_retries(5)
```

Questions?



Acknowledgements:

Many thanks to ND CMS group:

Prof. Kevin Lannon
Anna Woodard
Mathias Wolf
Kenyi Hurtado

btovar@nd.edu

http://ccl.cse.nd.edu/community/forum

http://ccl.cse.nd.edu/workshop/2016

extra slides

Stand-alone monitor

```
resource_monitor -L"cores: 4" -L"memory: 4096" -- matlab
```

```
cclws16  ~ > resource_monitor -i1 -Omon --no-pprint -- /bin/date
Thu May 12 20:27:21 EDT 2016
cclws16  ~ > cat mon.summary
{"executable_type":"dynamic","monitor_version":"6.0.0.9edd8e96","host":"cclws16.cse.nd.edu
","command":"/bin/date","exit_status":0,"exit_type":"normal","start":[1463099241605723,"us
"],"end":[1463099243000239,"us"],"wall_time":[1.39452,"s"],"cpu_time":[0.002999,"s"],"core
s":[1,"cores"],"max_concurrent_processes":[1,"procs"],"total_processes":[1,"procs"],"memor
y":[1,"MB"],"virtual_memory":[107,"MB"],"swap_memory":[0,"MB"],"bytes_read":[0.0105429,"MB
"],"bytes_written":[0,"MB"],"bytes_received":[0,"MB"],"bytes_sent":[0,"MB"],"bandwidth":[0
,"Mbps"],"total_files":[90546,"files"],"disk":[11659,"MB"],"peak_times":{"units":"s","cpu_
time":1.39452,"cores":0.394445,"max_concurrent_processes":0.394445,"memory":0.394445,"virt
ual_memory":1.39428,"bytes_read":1.39428,"total_files":1.39428,"disk":1.39428}}%
cclws16  ~ >
```

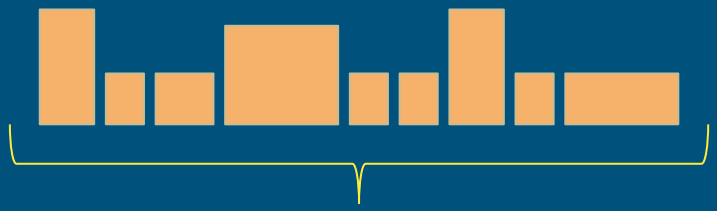
(does not work as well on static executables that fork)

Stand-alone monitor -- time series

```
% resource_monitor -Ooutput --with-time-series -- matlab

% tail -f output.series
```

(does not work as well on static executables that fork)

Tasks with Unknown Resource Requirements

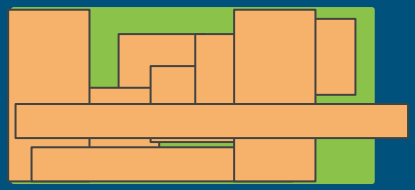


Tasks which size
(e.g., cores, memory, and disk)
is not known until runtime.

Available workers

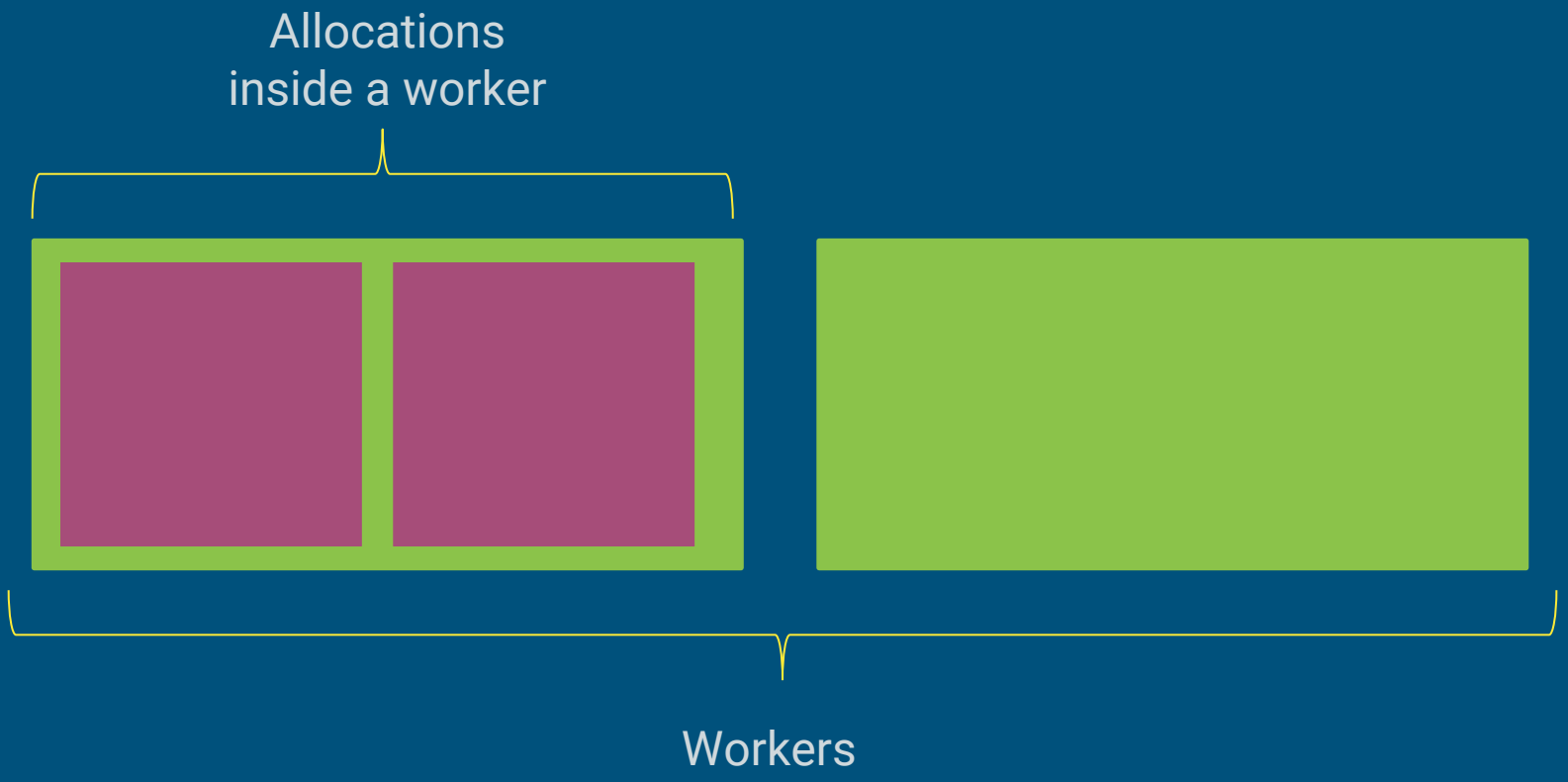
One task per worker:
Wasted resources, reduced throughput.

Many tasks per worker:
Resource contention/exhaustion, reduce throughput

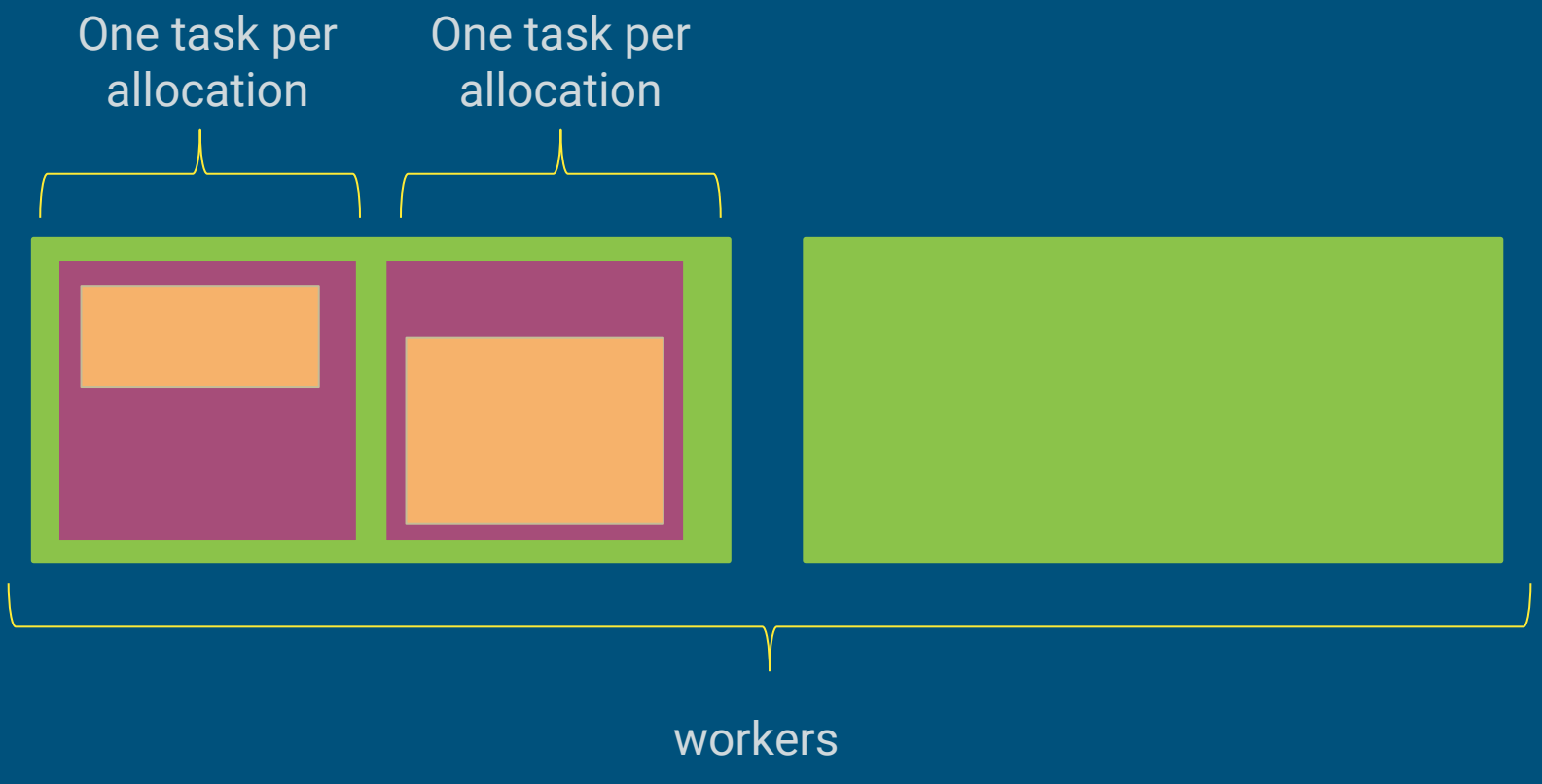


Task-in-the-Box

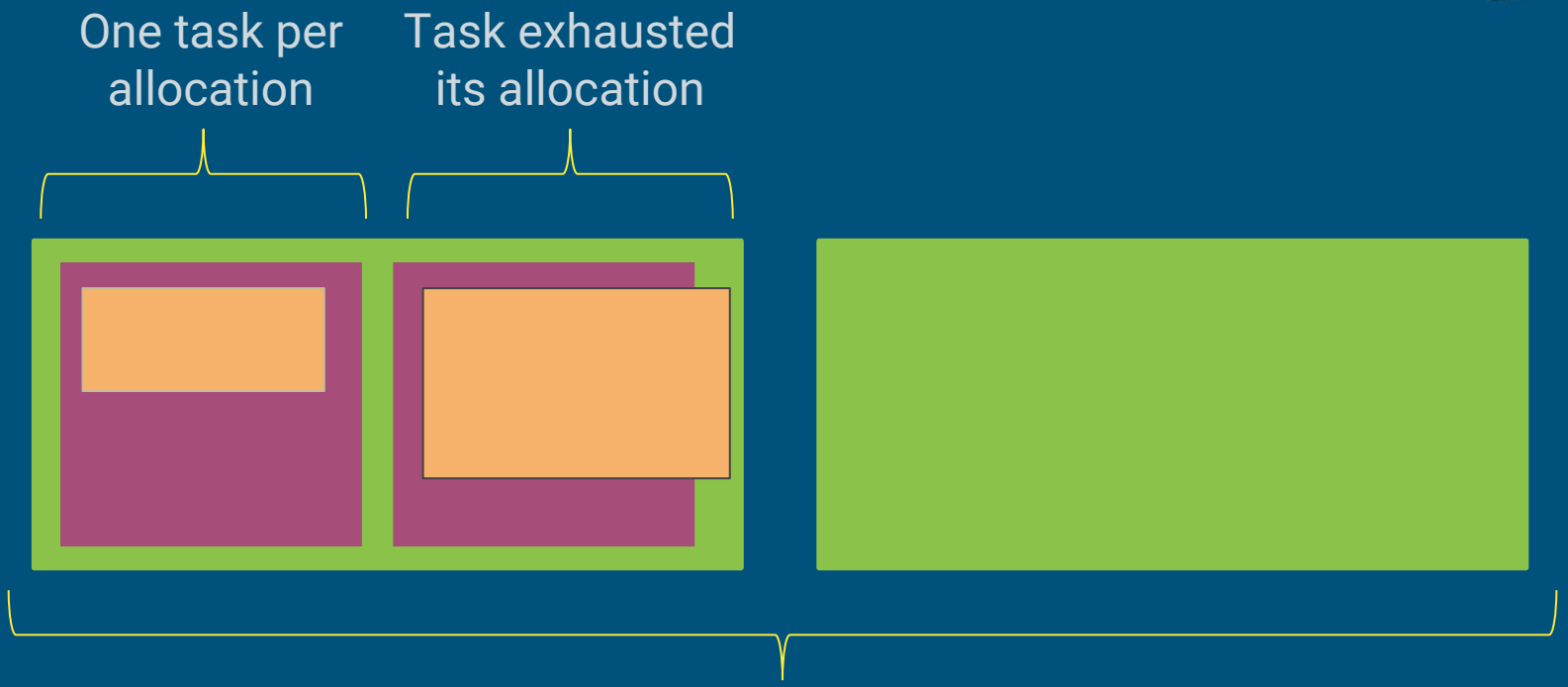
Task-in-the-Box



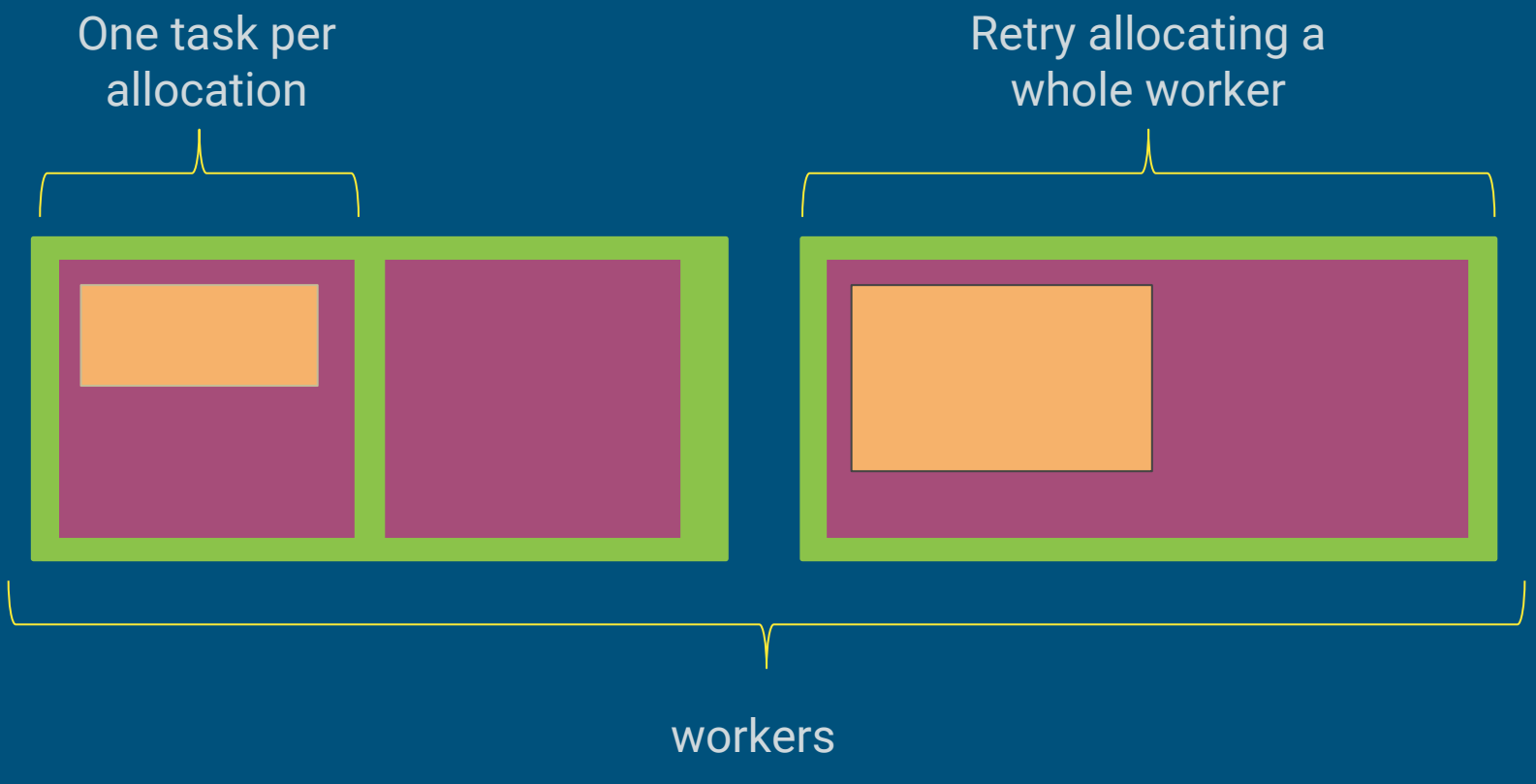
Task-in-the-Box



Task-in-the-Box



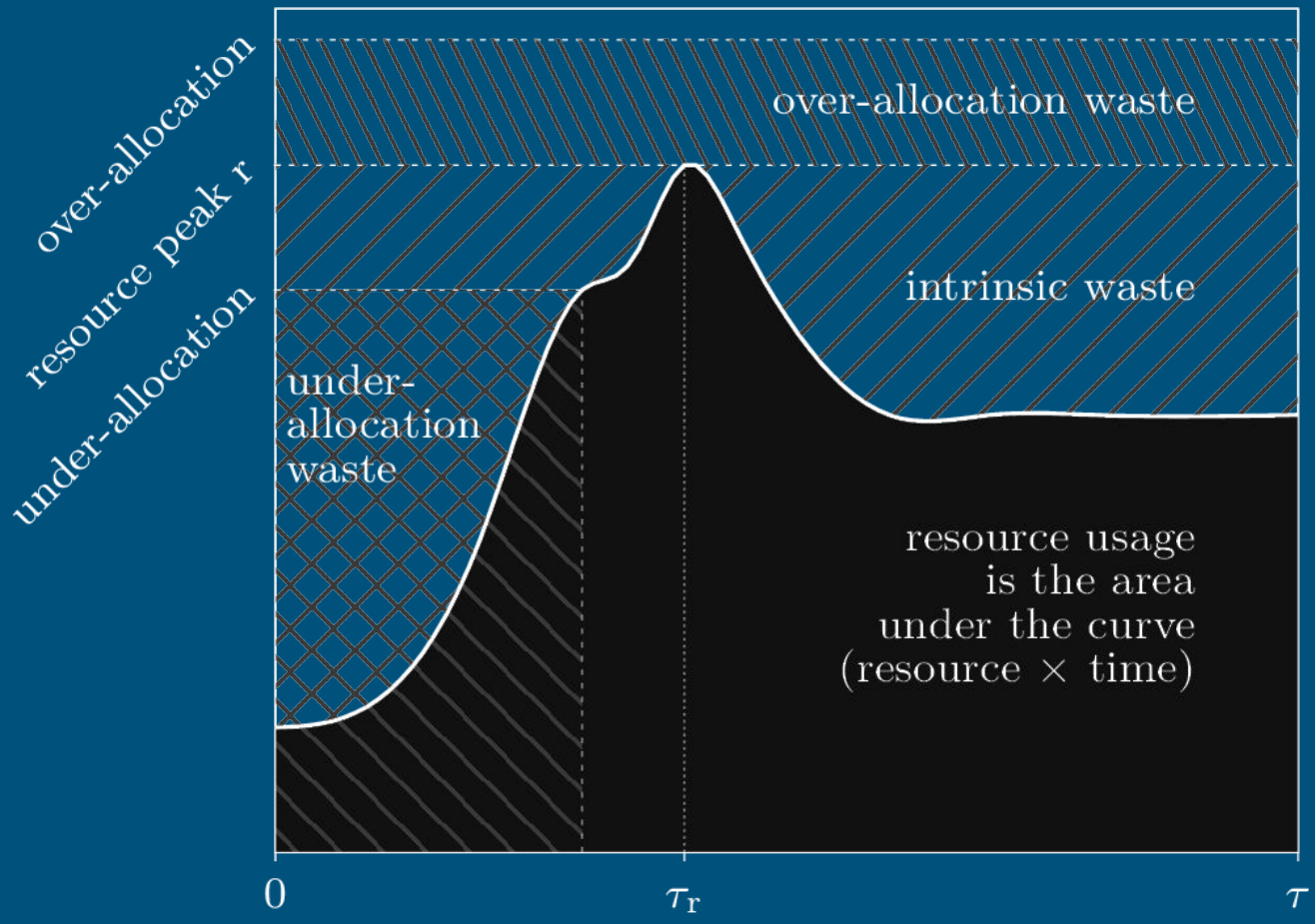
Task-in-the-Box



Main Challenge

What is a good allocation size?

Slow-peaks model



Random variables to describe usage:
Time to completion.
Size of max peak

Resource usage:
time x peak

Slow-peaks:
Resource peaks at the end of execution (conservative assumption)

Slow-peaks model

$$\mathrm{E}[\mathrm{waste}(r,\tau,a_1)] = \int_0^\infty \left(\underbrace{\int_0^{a_1} (a_1 - r)\tau p(r,\tau)\mathrm{d}r}_{\text{first allocation succeds}} + \underbrace{\int_{a_1}^{a_m} ((a_m + a_1 - r)\tau p(r,\tau)\mathrm{d}r}_{\text{final allocation succeds}} \right) \mathrm{d}\tau$$

$$= a_1 \underbrace{\int_{a_1}^{a_m} \int_0^\infty \tau p(r,\tau)\mathrm{d}\tau\mathrm{d}r}_{\text{mean wall-time for all tasks}}$$

$$+ a_m \int_{a_1}^{a_m} \underbrace{\int_0^\infty \tau p(\tau|r)\mathrm{d}\tau}_{\text{mean wall-time taks w. peak } r} \quad p(r)\mathrm{d}r$$

$$- \underbrace{\int_0^\infty \int_0^\infty r\tau p(r,\tau)\mathrm{d}\tau\mathrm{d}r}_{\text{used resources}},$$

Choice of:
maximum throughput
minimum waste.

Optimizations over expectations

O(n) simple arithmetic expressions that use only information available during execution.