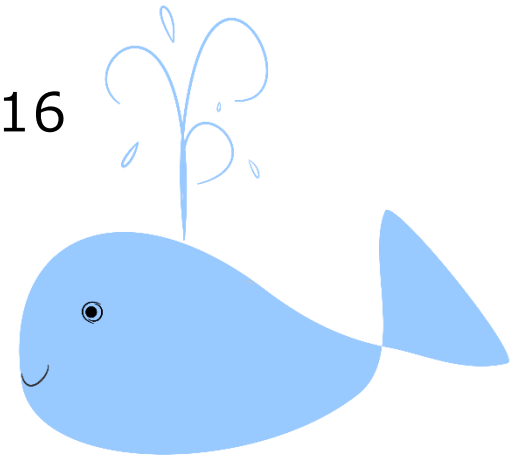
Parallelizing CI using Docker Swarm-Mode

Akihiro Suda <suda.akihiro@lab.ntt.co.jp>
NTT Software Innovation Center

Who am I

https://github.com/AkihiroSuda

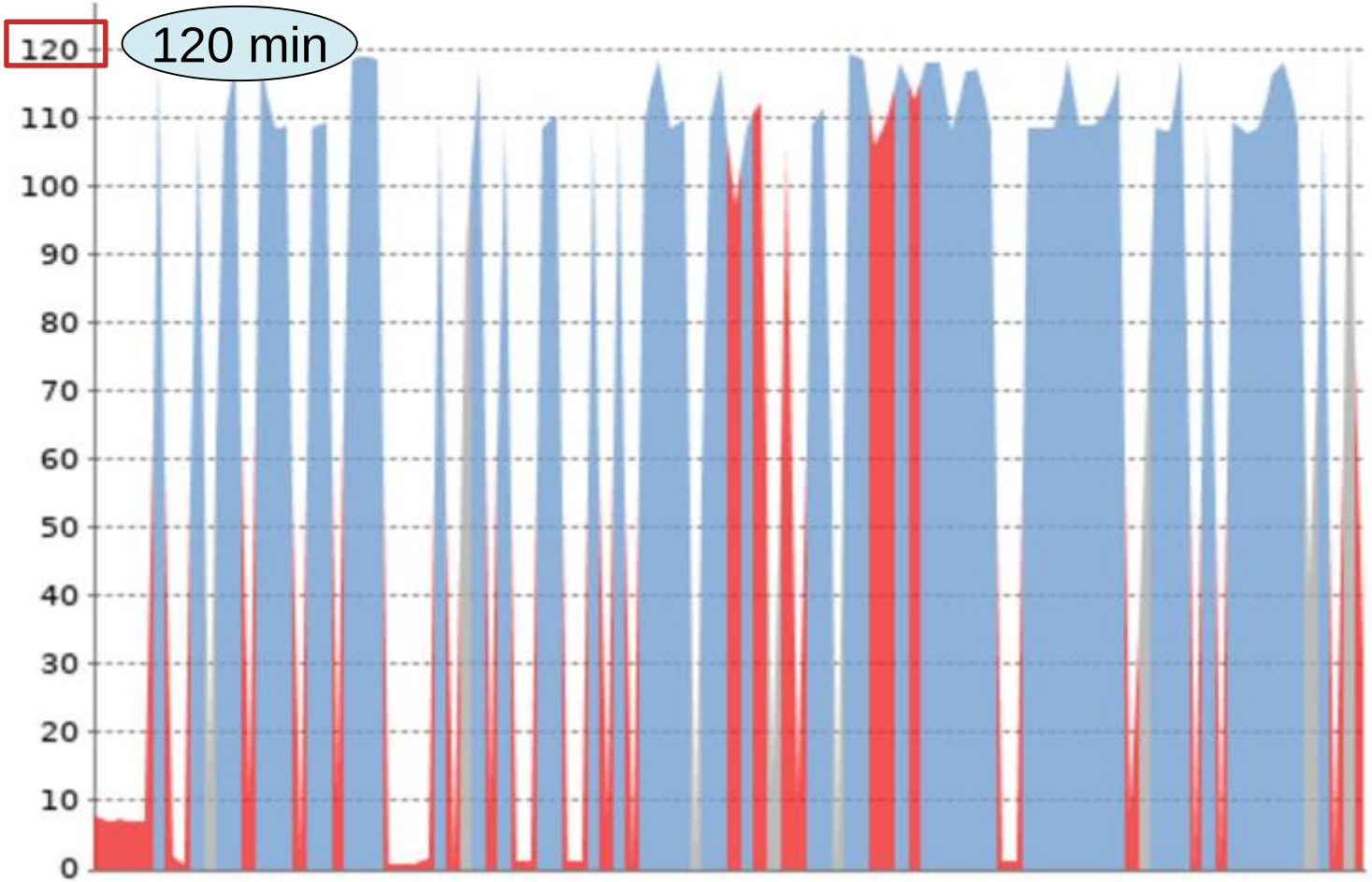
- **Software Engineer at NTT Corporation**
- **Docker core maintainer**
- **Previous talks at FLOSS community**
  - FOSDEM 2016
  - ApacheCon Core North America 2016
  - …



A problem in Docker project: CI is slow

capture: March 3, 2017

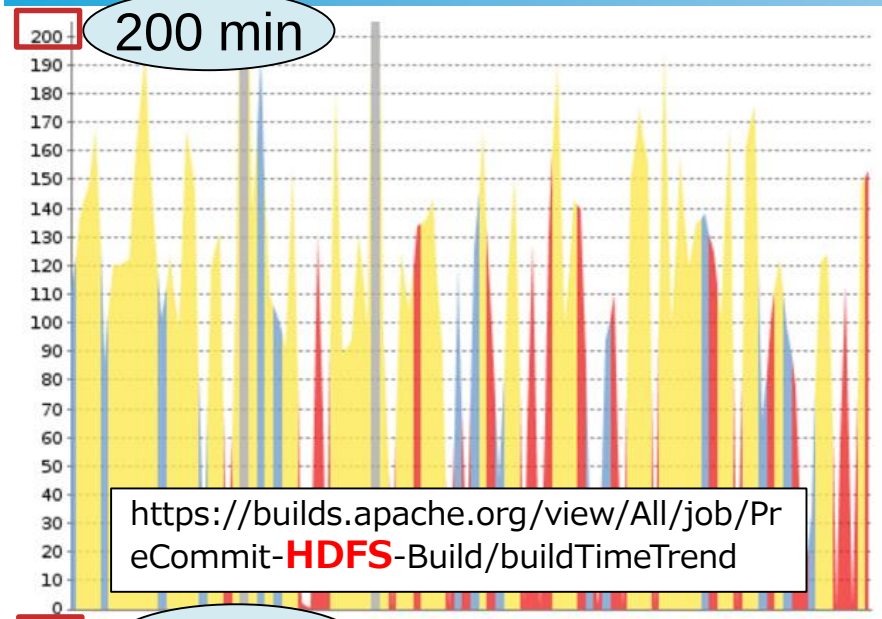
https://jenkins.dockerproject.org/job/Docker-PRs/buildTimeTrend

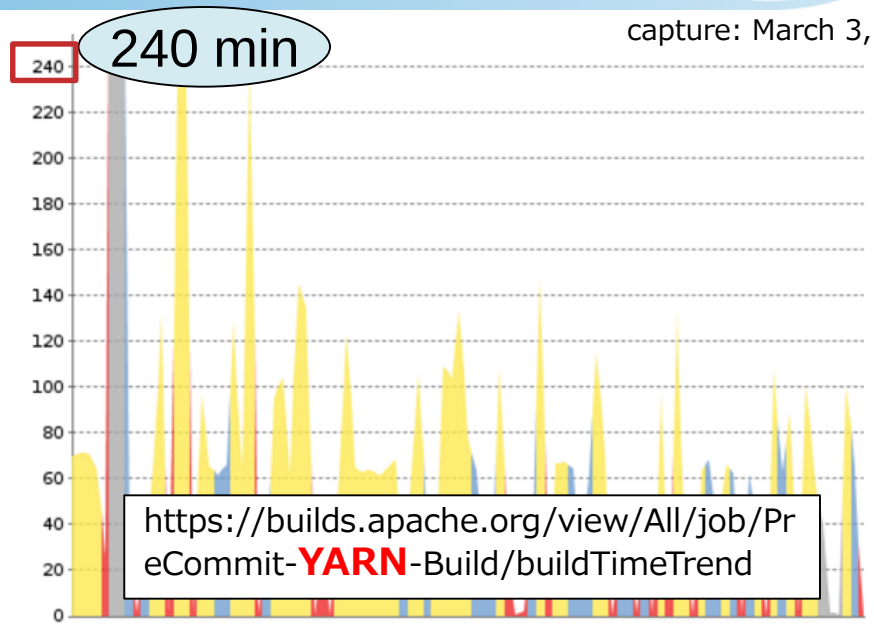


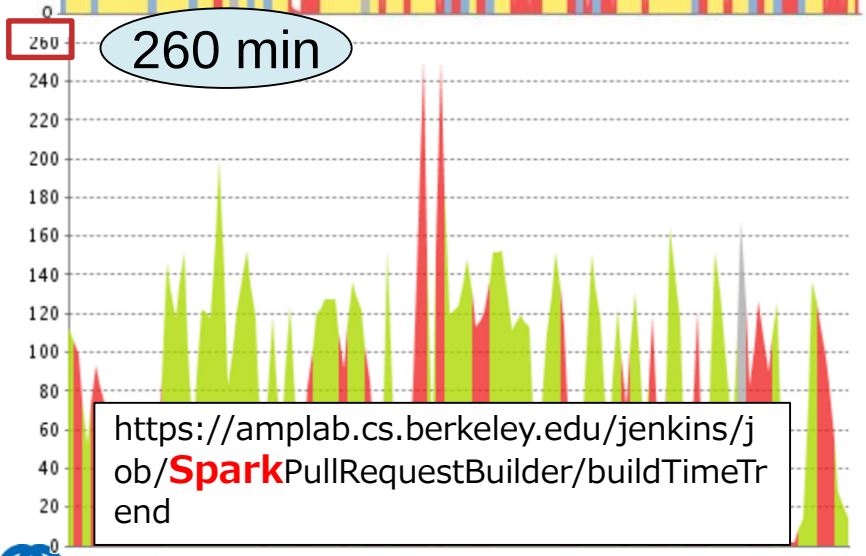
How about other FLOSS projects?

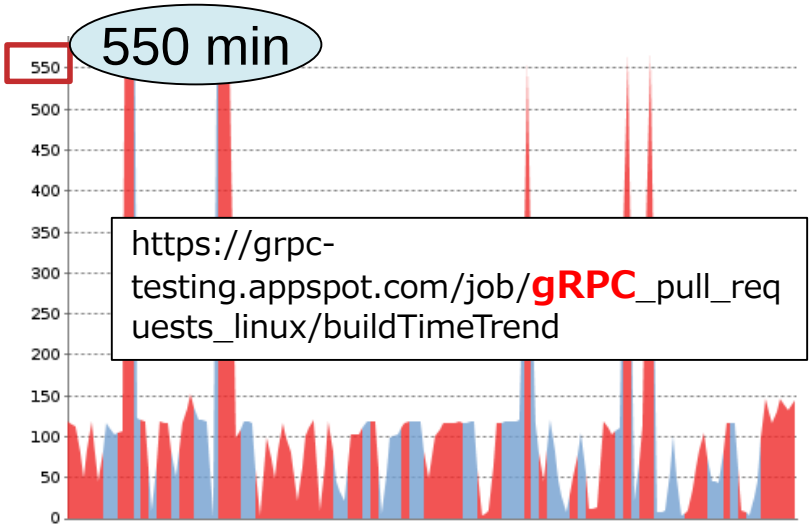
Innovative R&D by NTT

capture: March 3, 2017







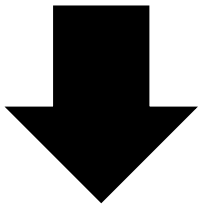


NTT

(for ease of visualization, picked up some projects that use Jenkins from various categories)

Why slow CI matters?

- **Blocker for reviewing/merging patches**
- **Discourages developers from writing tests**
- **Discourages developers from enabling additional testing features (e.g. race detector)**

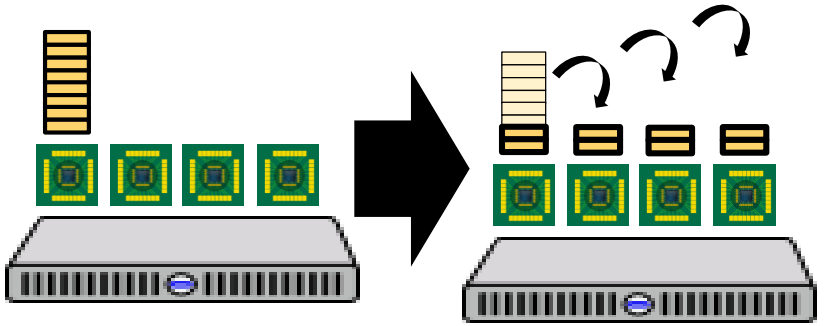


Poor implementation quality &
Slow development cycle

Solution: Parallelize CI ?

e.g.

- `go test -parallel N`
- `mvn --threads N`
- `parallel`
- ...



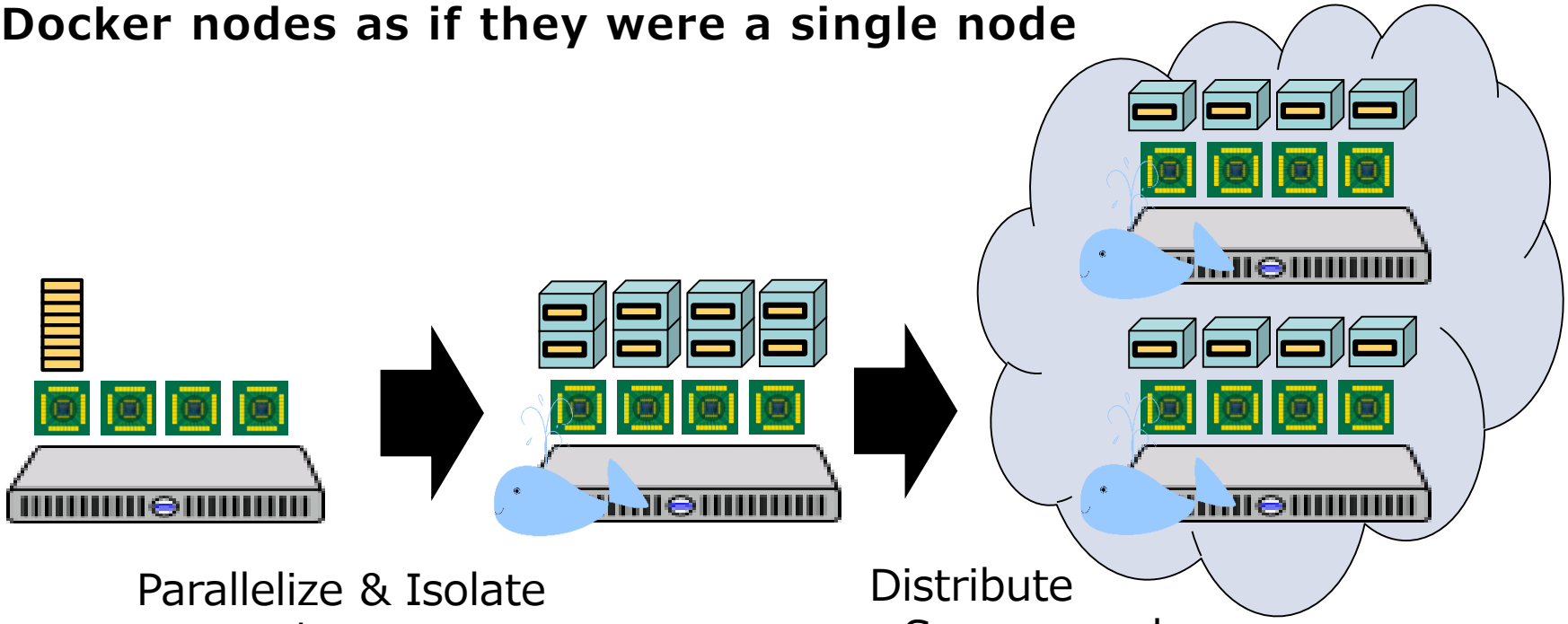
But parallelization is not enough ☹

- **No isolation**
  - Concurrent test tasks may race for certain shared resources (e.g. files under `/tmp`, TCP port, ...)
- **Poor scalability**
  - CPU/RAM resource limitation
  - I/O parallelism limitation

Solution: Parallelize CI across Docker Swarm-mode

- **Docker provides isolation**

- **Docker Swarm-mode provides scalability, plus allows you to manage multiple Docker nodes as if they were a single node**



Parallelize & Isolate
using Docker containers

✓Isolation

Distribute
across Swarm-mode

✓Scalability

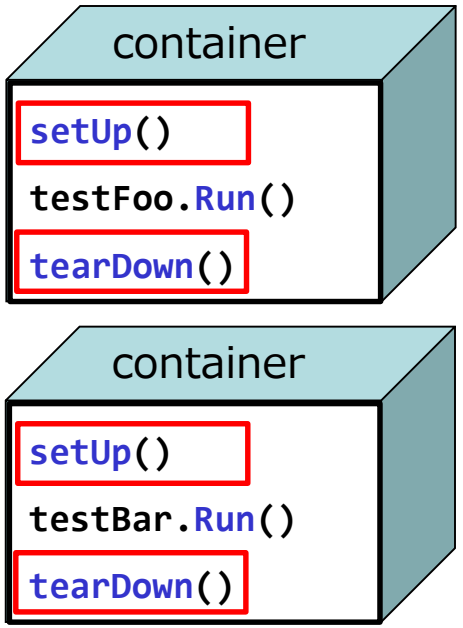
Chunking

- For ideal isolation, each of the test functions should be encapsulated into independent containers

- But this is not optimal in actual due to setup/teardown code

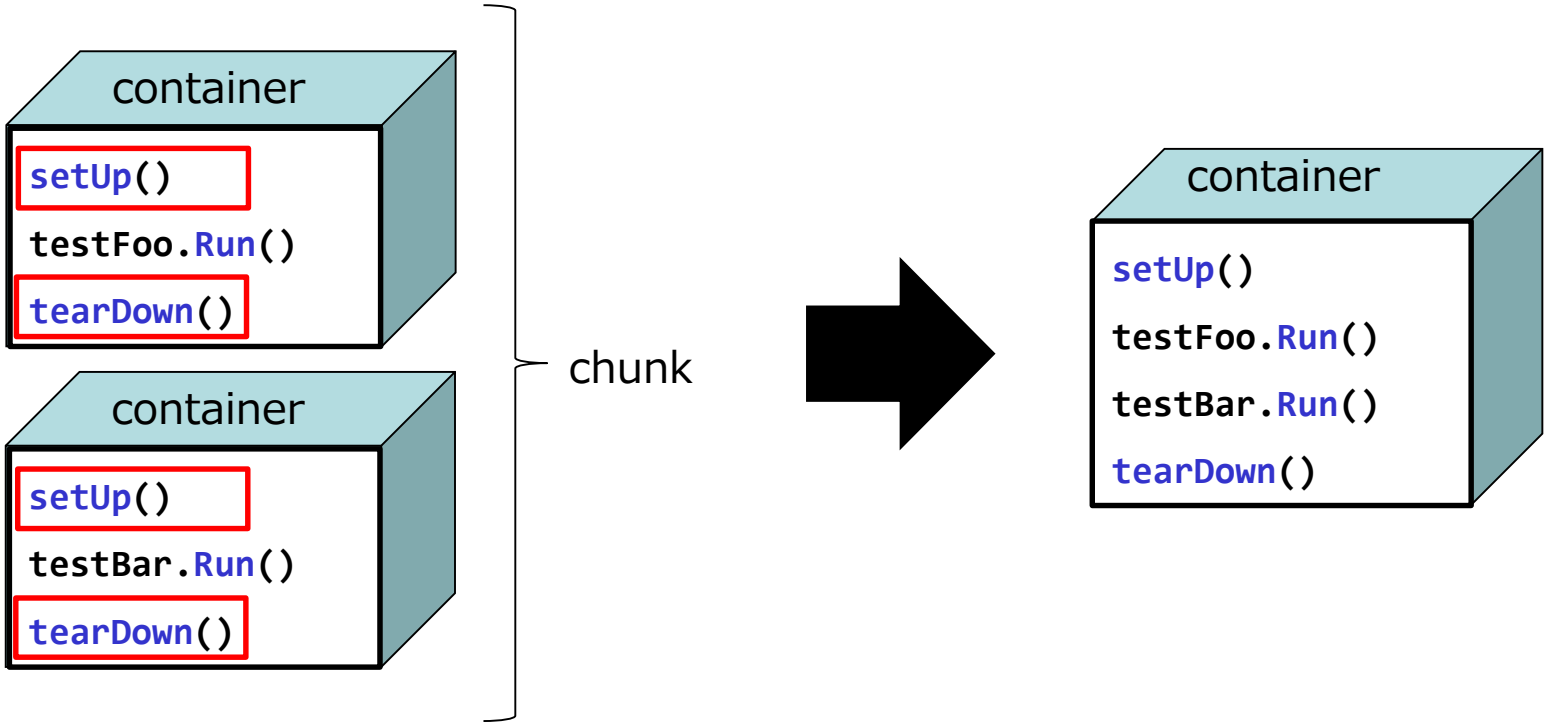
```
func TestMain(m *testing.M) {
    setUp()
    m.Run()
    tearDown()
}
func TestFoo(t *testing.T)
func TestBar(t *testing.T)
```

redundantly executed



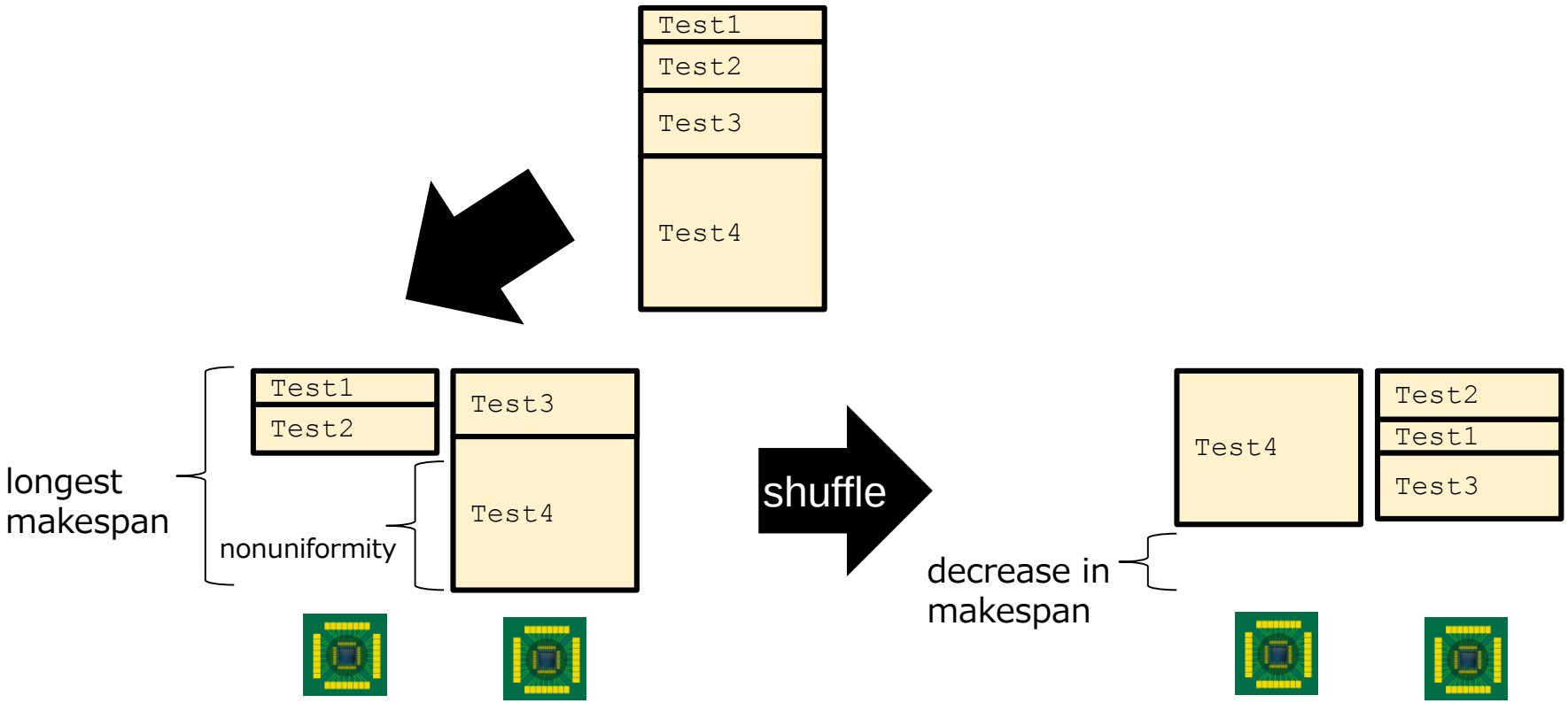
Chunking

- **So we execute a chunk of multiple test functions sequentially in a single container**



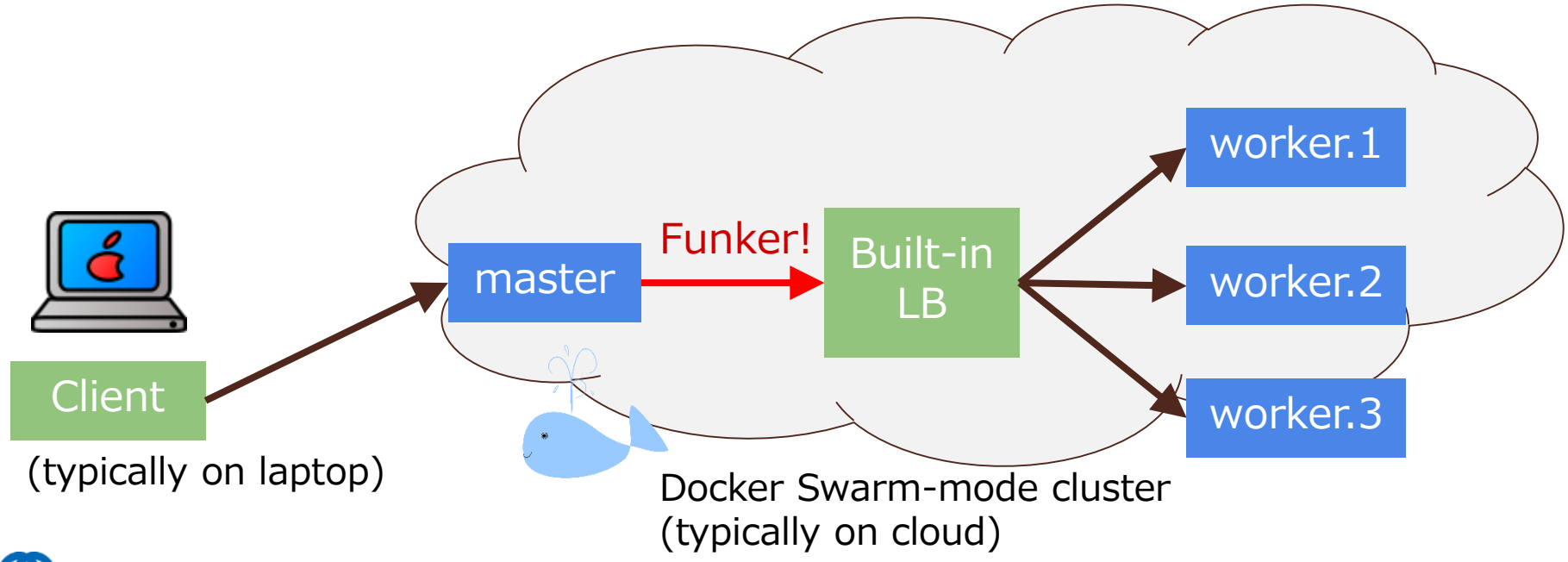
Shuffling

- **makespans of the chunks are not uniform**
- **So we shuffle the chunks for "hope" of optimal scheduling**
  - No guarantee for optimal schedule



Implementation

- **We use Funker (github.com/bfirsh/funker)**
  - FaaS-like architecture
  - Loads are automatically balanced via Docker's built-in LB
  - No explicit task queue
  - No `docker exec` hack
  - Could be easily portable to other container orchestrators



Experimental result

- **Evaluated my hack against the CI of Docker itself**
  - Of course, this hack can be applicable to CI of other software as well
- **Target: Docker 16.04-dev (git:7fb83eb7)**
  - Contains 1,648 test functions
- **Machine: Google Computing Engine n1-standard-4 instances (4 vCPUS, 15GB RAM)**

Time: average of 5 runs

Detailed result

Number of containers running in parallel
(chunk size is inversely proportional to this)

Nodes	10	30	50	70
1	**15m3s**		N/A (more than 30m)	
2	12m7s	**10m12s**	11m25s	13m57s
5	10m16s	6m18s	**5m46s**	6m28s
10	8m26s	4m31s	**4m10s**	4m20s

Even with a single node,
the result is significantly better than
the traditional testing (1h22m7s)

Fastest configuration

The code is available!

Innovative R&D by NTT

PR (merged):
https://github.com/docker/docker/pull/29775

```
$ cd $GOPATH/src/github.com/docker/docker
$ make build-integration-cli-on-swarm
$ ./hack/integration-cli-on-swarm/integration-cli-on-swarm \
  -replicas 50 \
  -shuffle \
  -push-worker-image your-docker-registry.example.com/worker
```

NTT

Is it applicable to other software as well?

- Yes, with just a few line of modifications
- Planning to split this hack into an independent generic test tool

Future work

- "Shuffling" does not always result in optimal schedule

- Future work: learn the past execution history to optimize the schedule

Recap

- **Docker Swarm-mode is effective for parallelizing CI jobs**

- **Some hacks for optimizing schedule**