Putting the “Micro” Back in Microservices

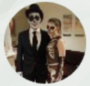
Sol Boucher, *Carnegie Mellon University*

Joint work with:
Anuj Kalia
David G. Andersen
Michael Kaminsky, *Intel Labs*

The Many Potential Benefits of Serverless Computing

The emerging cloud service can reduce costs and speed deployment times

Wall Street Journal



Elliot Forbes Follow
Professional Software Developer.
Dec 31, 2017 · 4 min read

How Serverless Computing will Change the World in 2018

Hacker Noon

By Tim Anderson 11 Jul 2016 at 09:38 14 SHARE ▾

Dr Werner Vogels, Amazon CTO, speaking in London

AWS Summit London Amazon CTO Dr Werner Vogels talked up the value of serverless computing at the AWS (Amazon Web Services) London Summit last week.

The Register

EVALUATE

Serverless computing is the next big thing -- and it's already here

Tech Target

INSIDER

How AWS will own you through serverless computing

Billions of dollars invested in servers and software for serverless computing

InfoWorld

The hope for serverless computing

Only have to manage **code**

Microservices invoked by **triggers**

Microservices are **stateless**

This makes the system **scalable**

Fine-grained billing that **scales to zero**

Goal: **Reduce microservice invocation latency**

Median AWS Lambda warm-start latency **25 ms**

Median cold-start latency **>160 ms** [Yesterday, ATC'18]

Latency between Azure VMs **~10 *μs*** [AccelNet, NSDI'18]

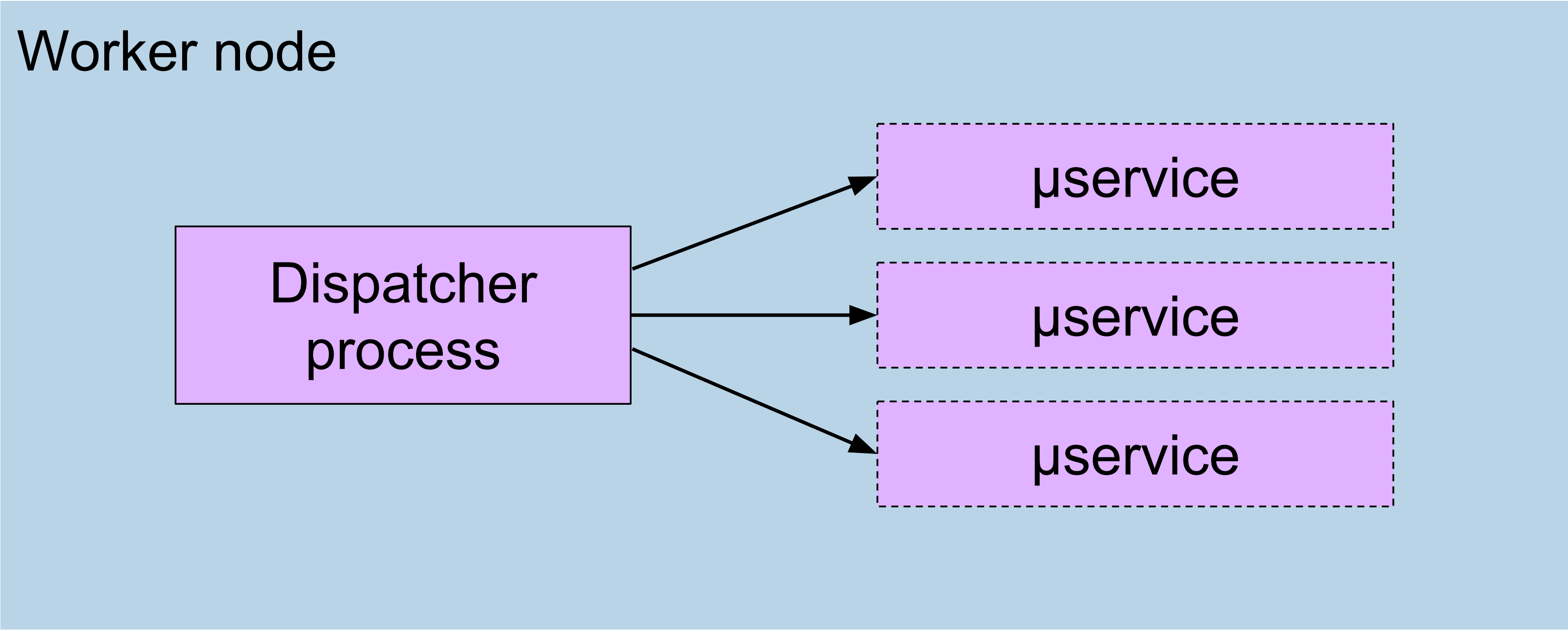
Commit ACID transactions in **~20 *μs*** [FaRM, SOSP'15]

Speed begets generality

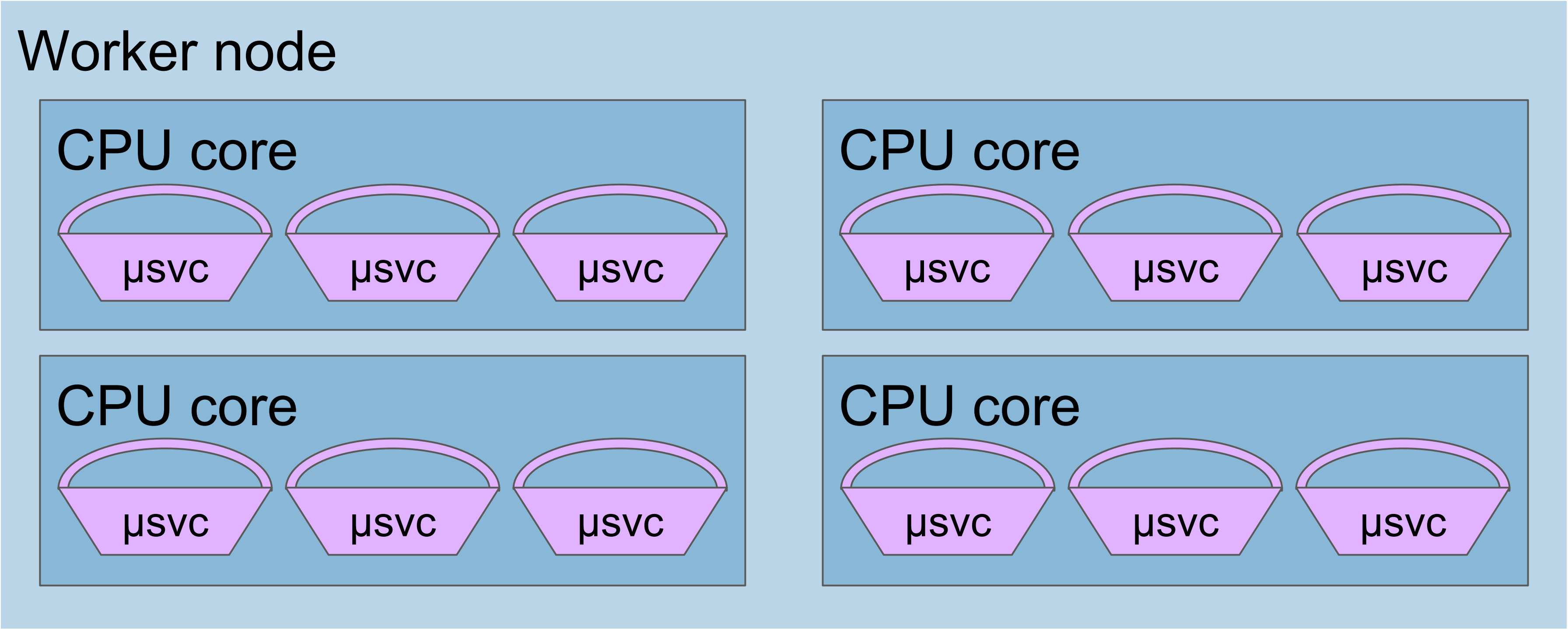
"Make it fast, rather than general or powerful."

— *Butler Lampson*

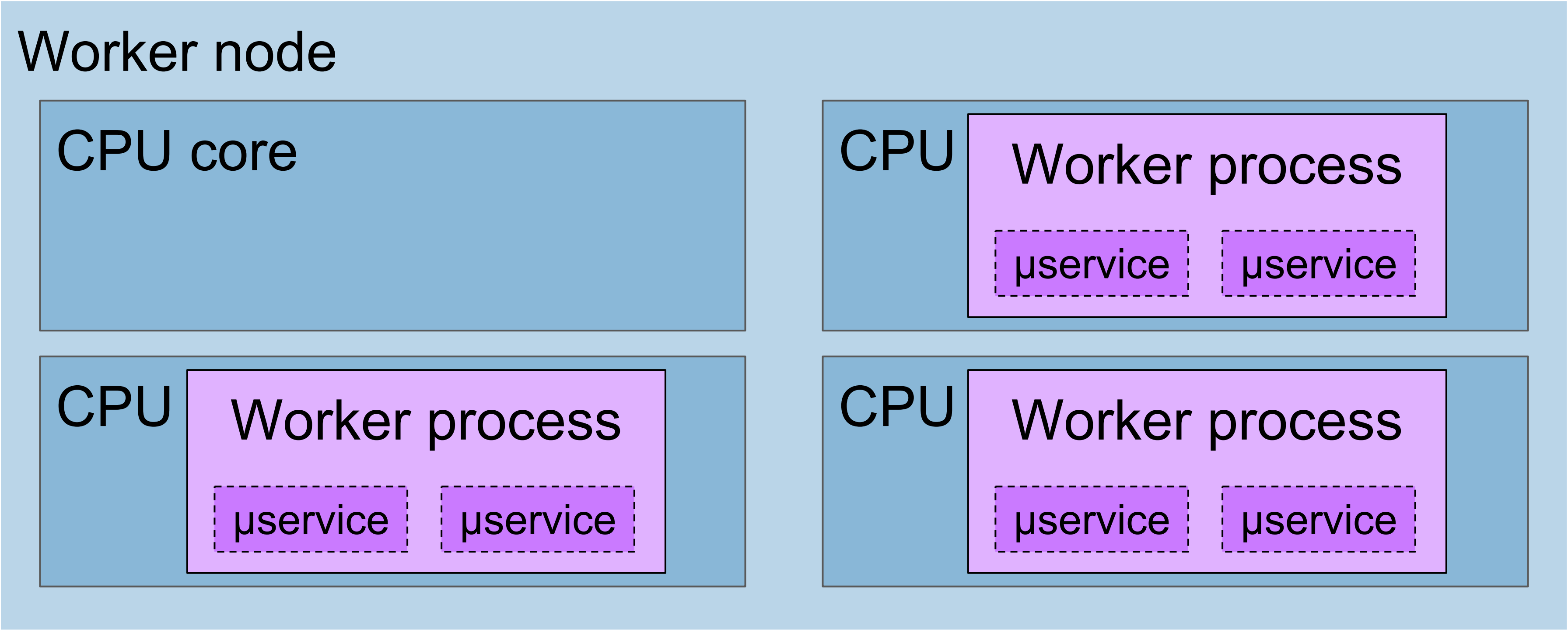
Current request path



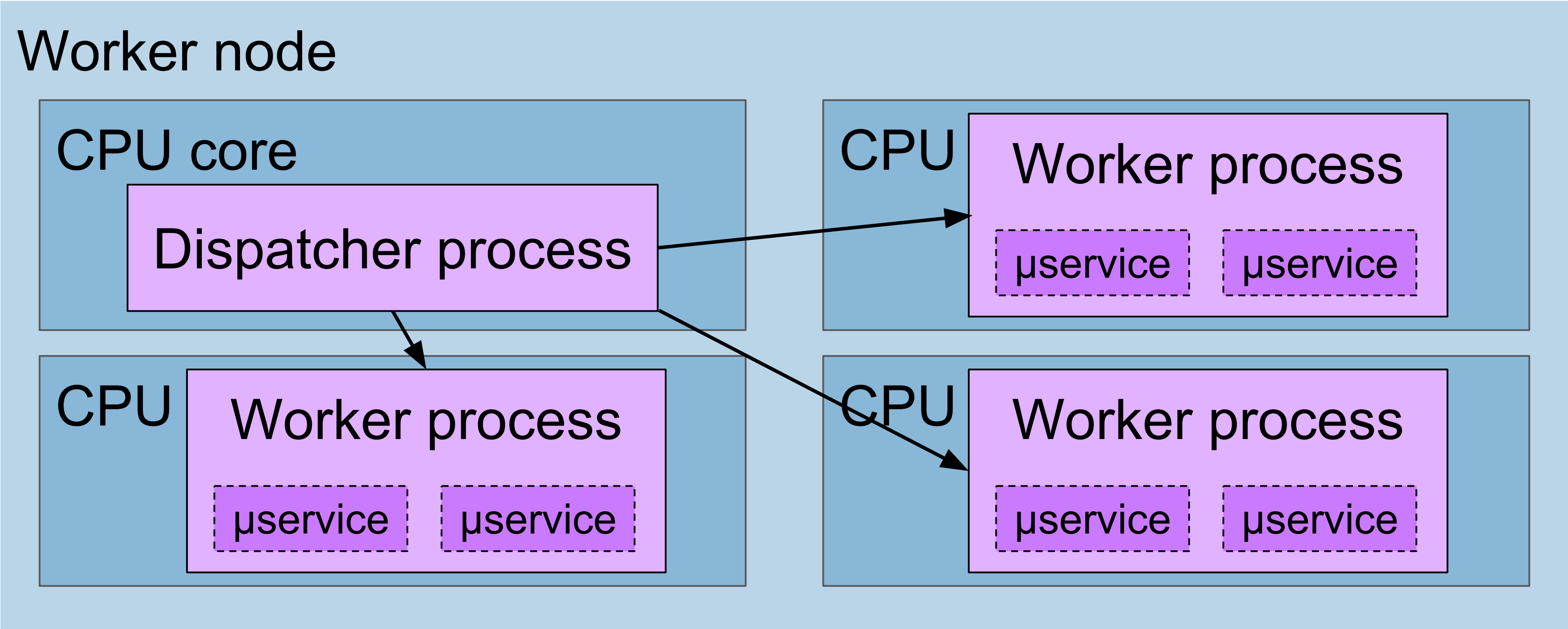
Proposal: **Reduce overhead...**



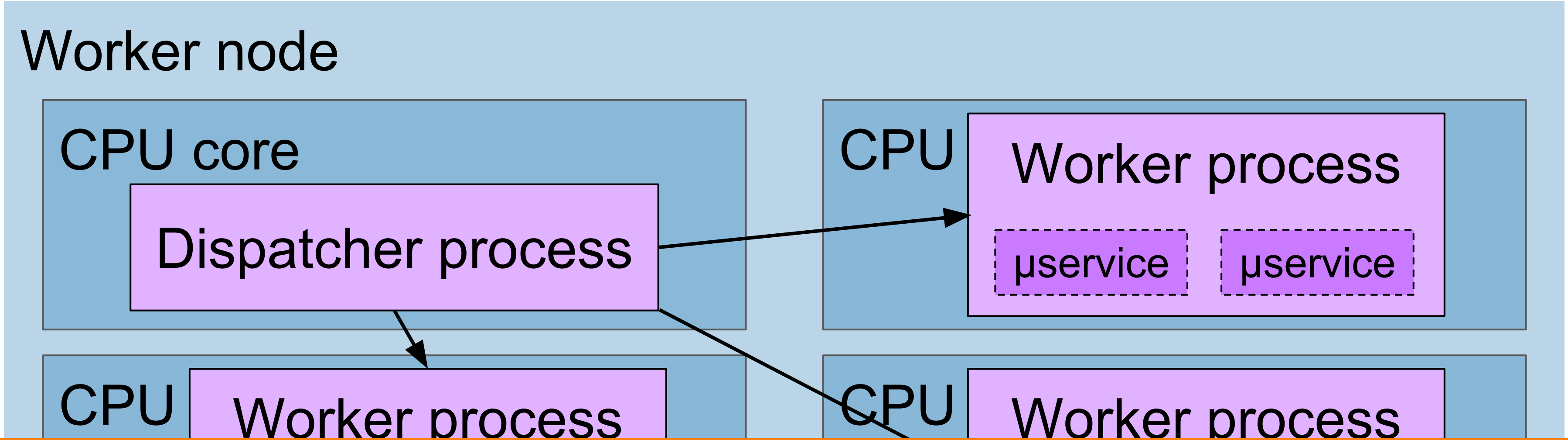
Proposal: **...by running code in shared workers...**



Proposal: **...and distributing work using polling**



Proposal: **...and distributing work using polling**



But...

How do we provide isolation?

How do we achieve isolation similar to processes?

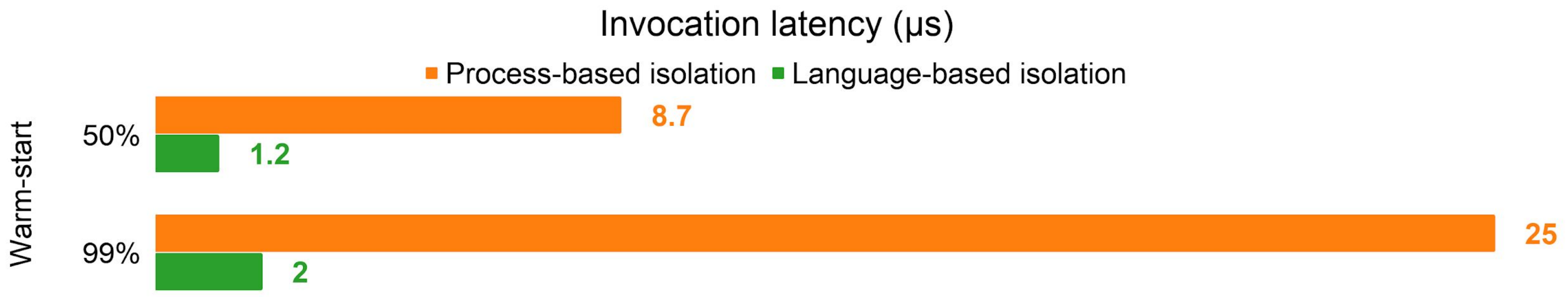
Language-based isolation: compile-time safety guarantees

Fine-grained preemption: intra-process task interruption

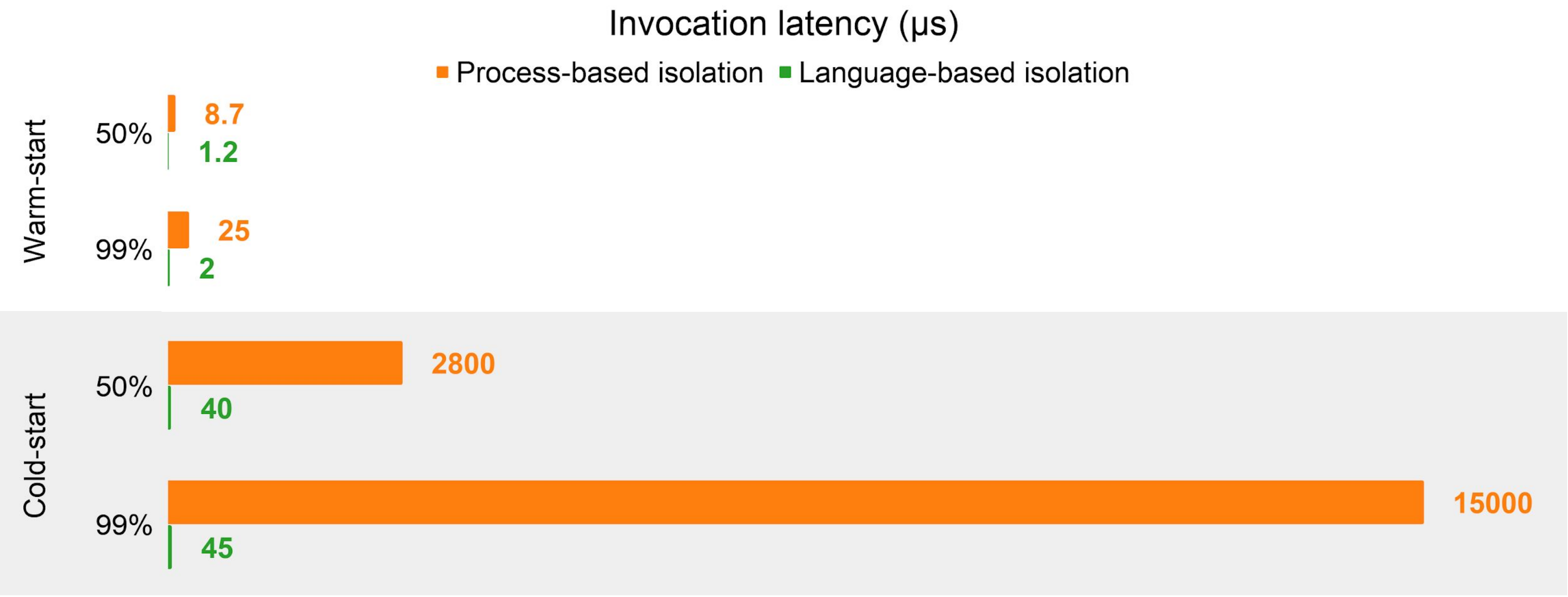
We use Rust for this, inspired by NetBricks [OSDI'16] and Tock [SOSP'17]

User submits Rust code; we verify it

Language-based isolation cuts invocation latency



Language-based isolation cuts invocation latency



Language-based isolation: **Use Rust**

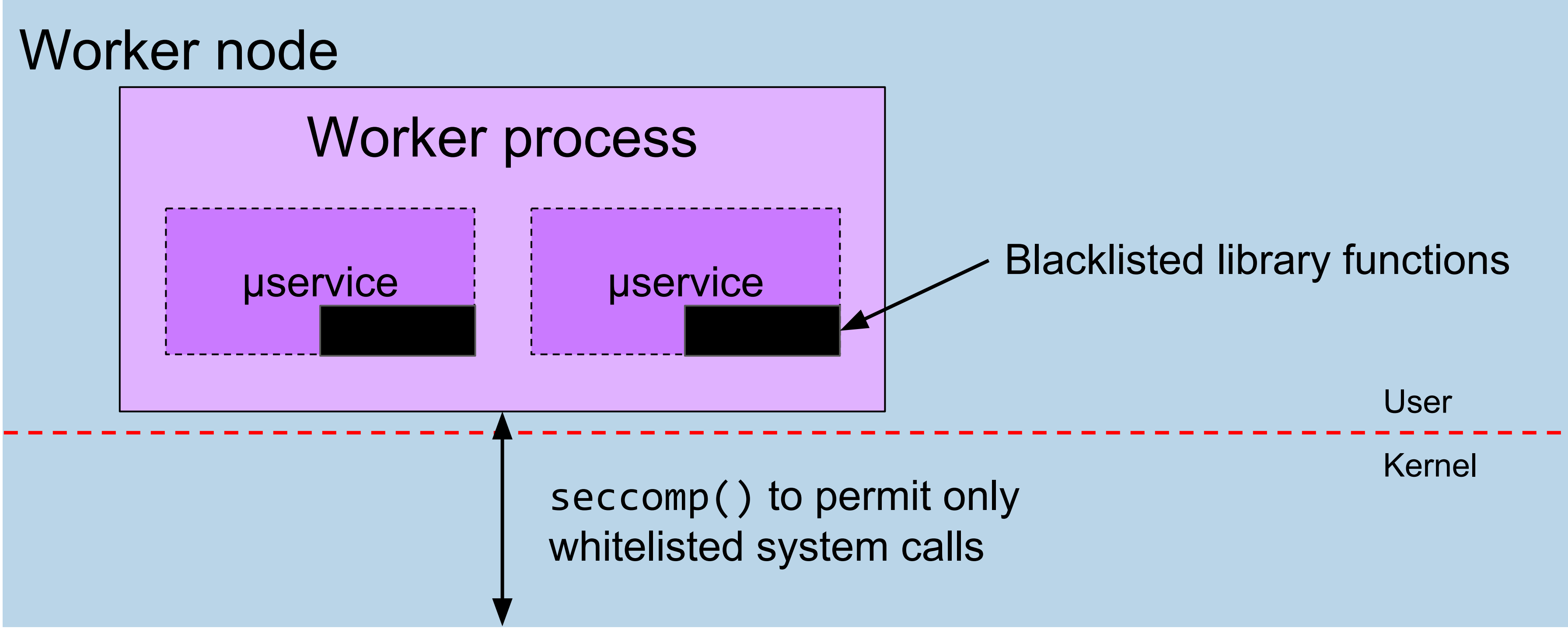
Rust is…

- Strongly typed, compiled
- Specified safe subset
- No garbage collector

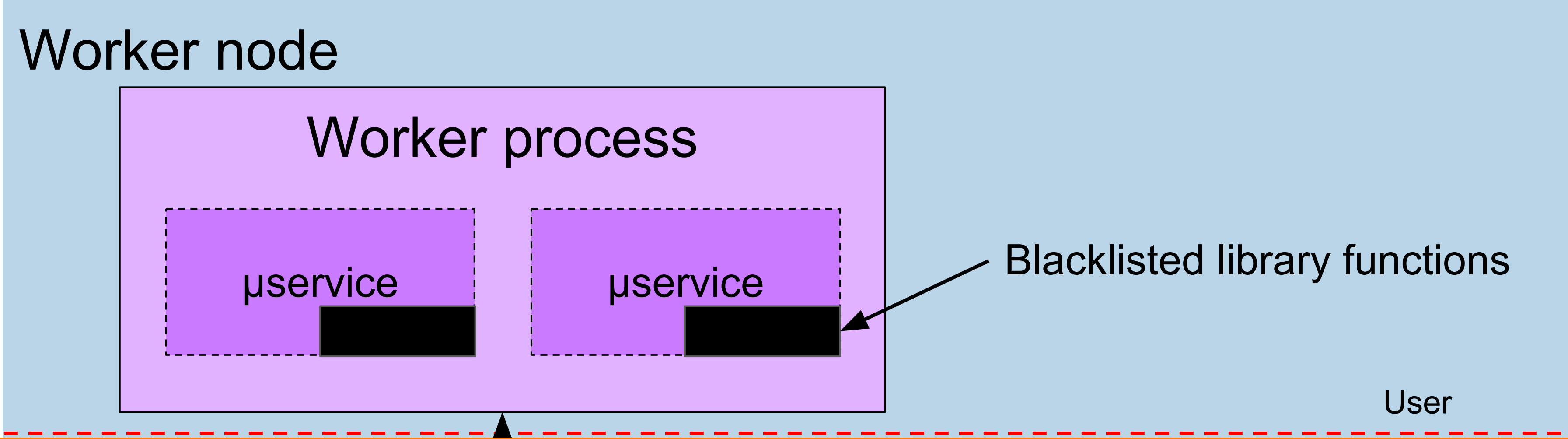
Memory safety guarantees:

- No dereferencing null/dangling pointers
- All variables initialized to valid values
- Enforced data immutability

Language-based isolation: **Defense in depth**



Language-based isolation: **Defense in depth**



But...

What if a microservice doesn’t yield?

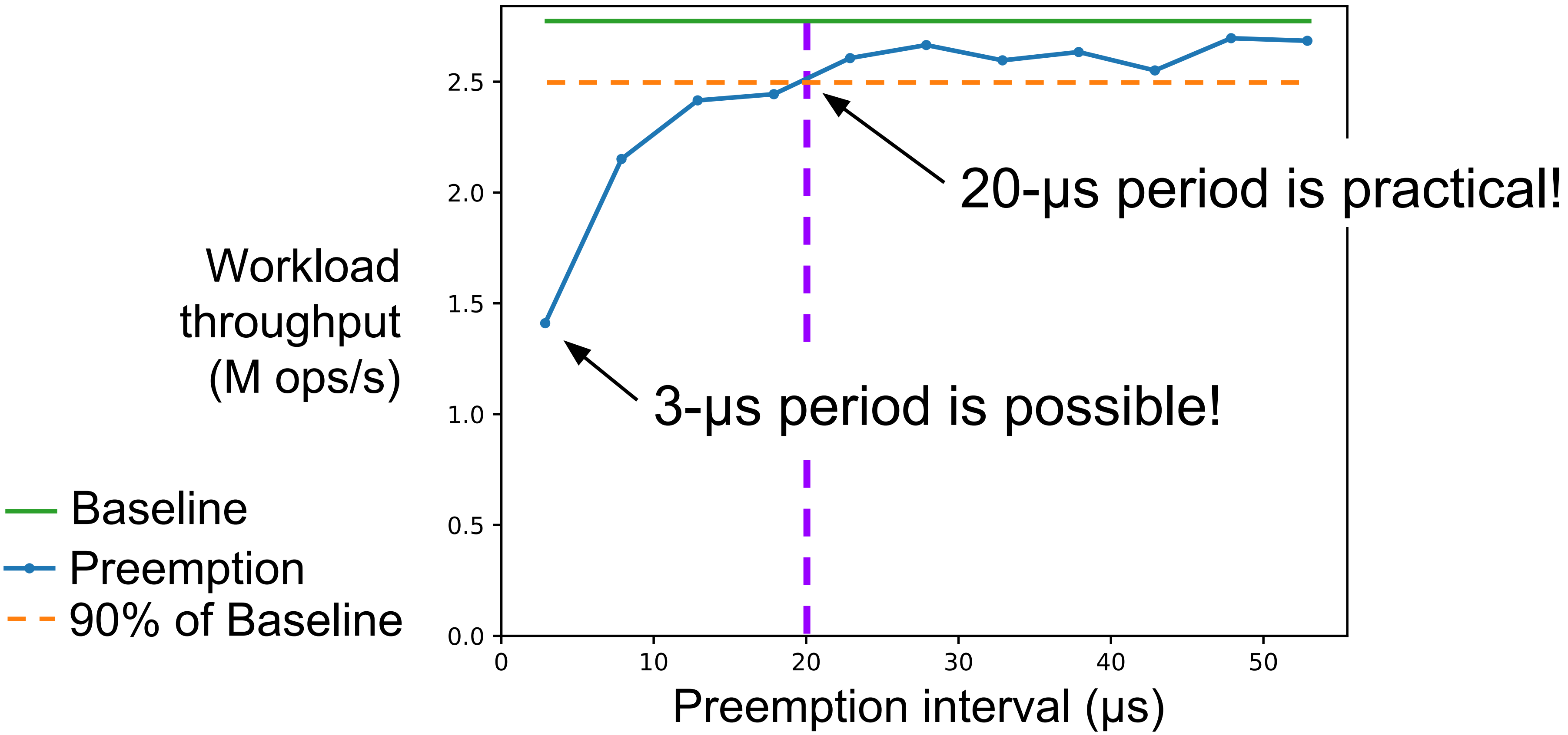
CPU timesharing: **Fine-grained preemption**

Goal: Recover from microservice that doesn’t return quickly

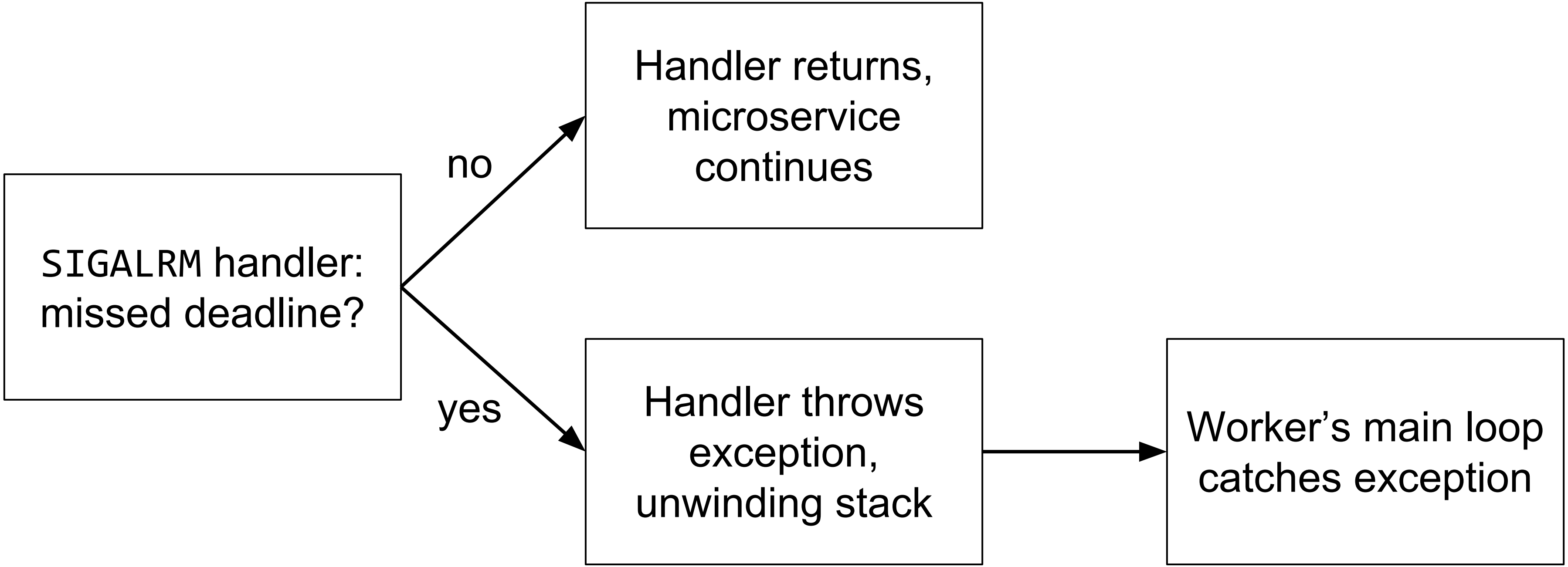
1. Regain control of the CPU
2. Abort/clean up after microservice’s code

Implementation: POSIX timers, special cleanup logic

Fine-grained preemption



Fine-grained preemption: **Aborting and cleanup**



Trust model

Trusted computing base:

- Rust compiler, standard library
- Any allowed `unsafe` or native dependencies

Successful *compilation* indicates microservice is memory safe

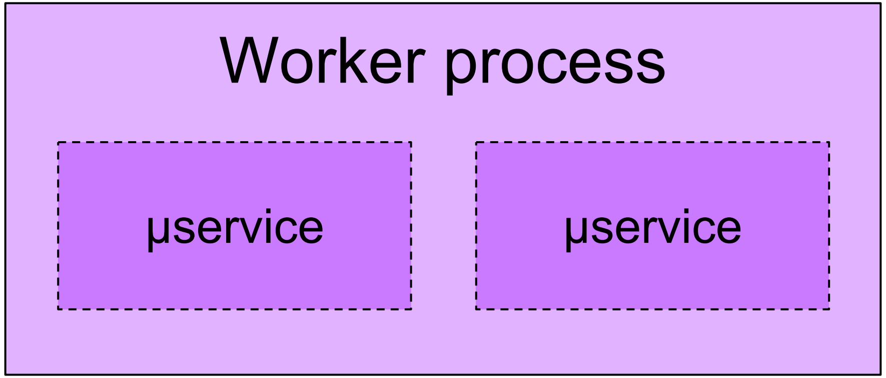
Successful *linking* indicates all dependencies are trusted

Recap

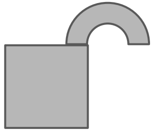
✓ Consolidate microservices into shared processes
✓ Improved local invocation latency by orders of magnitude
✓ (Hopefully) better resource utilization

→ Current limitations and future work

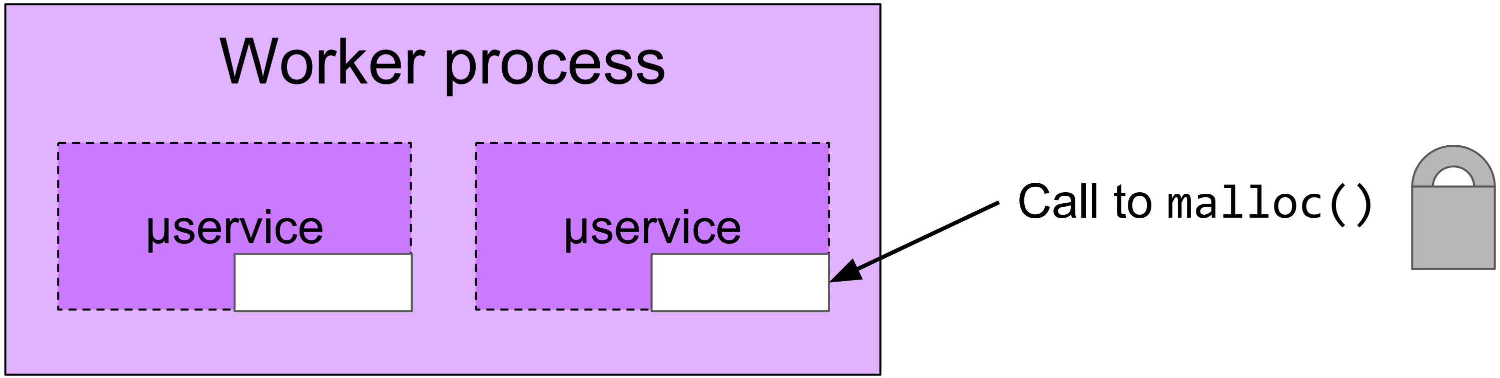
Future work: **Aborting/cleanup limitations**



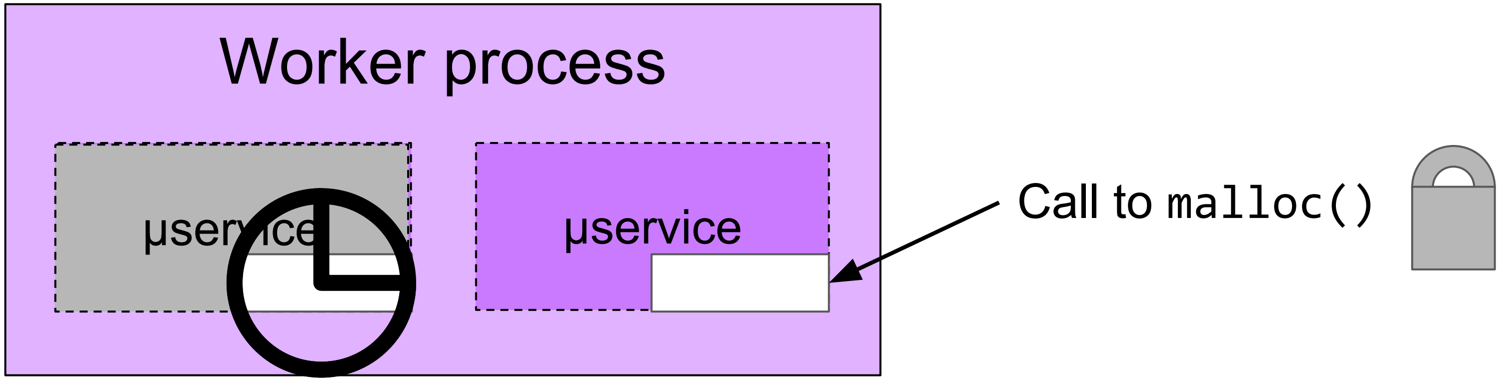
Call to `malloc()`



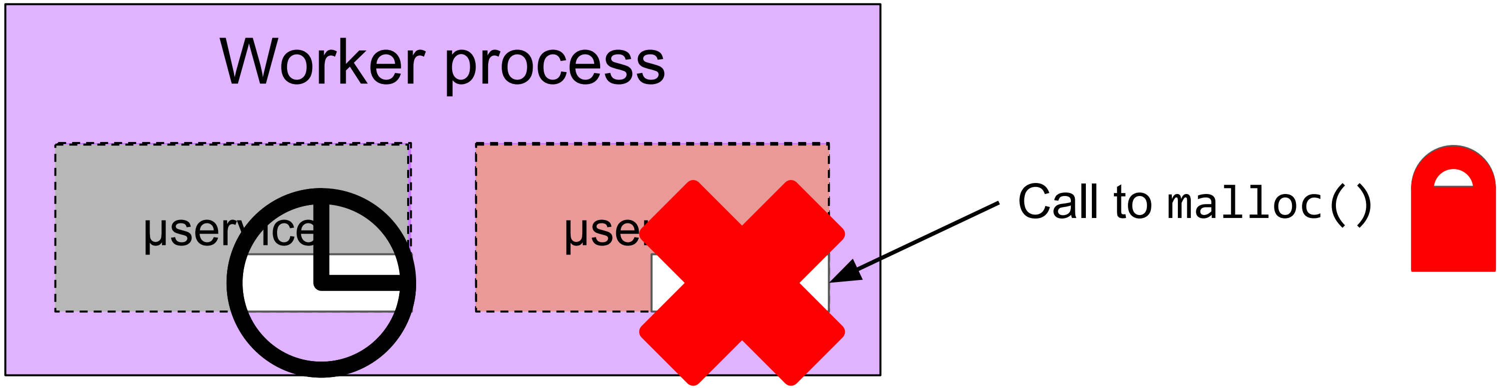
Future work: **Aborting/cleanup limitations**



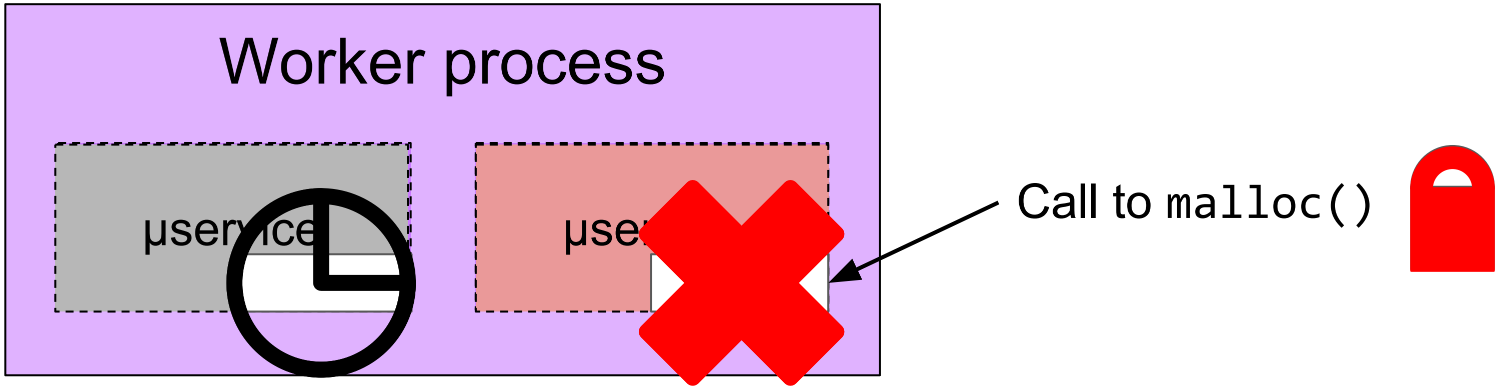
Future work: **Aborting/cleanup limitations**



Future work: **Aborting/cleanup limitations**



Future work: **Aborting/cleanup limitations**



Upcoming: More general accounting/deallocation scheme

- Operates outside the Rust runtime
- Disables preemption during trusted library routines

Future work: **Side-channel attacks**

Heightened Spectre vulnerability requires hardware mitigation

Must consider microservices' access to:

- Process's proximity to resource limits
- Addresses and timings from the dynamic allocator
- File descriptor numbers

Shorter microservice durations make behavior less obscure

Conclusion

Improved performance by shifting isolation abstraction layer

Replaced traditional process-based isolation with:

- **Language-based isolation**
- **Fine-grained preemption**

Conclusion

Improved performance by shifting isolation abstraction layer

Replaced traditional process-based isolation with:

- **Language-based isolation**
- **Fine-grained preemption**

Questions?

Thank you!