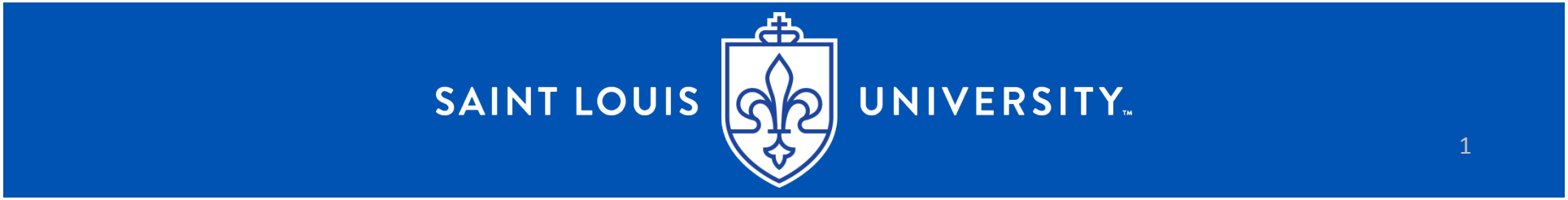
Incorporating Physical Dynamics Into Systems Mechanisms

NGOSCPS 2019 – Next Generation Operating Systems for Cyber Physical Systems

David Ferry
Saint Louis University



Two Directions of Innovation

1) Traditional OSes have already figured out scaling and multicore
   - Modern OSes are highly evolved, what lessons can we take away from them?
   - We conceptualize the traditional timer wheel algorithm to the problem of event detection.

2) CPSes have already figured out how to cleverly leverage system dynamics
   - More information shouldn't make things worse, can a dynamics-aware OS do better than a traditional OS?
   - We consider Read-Copy-Update synchronization in the context of dynamics.

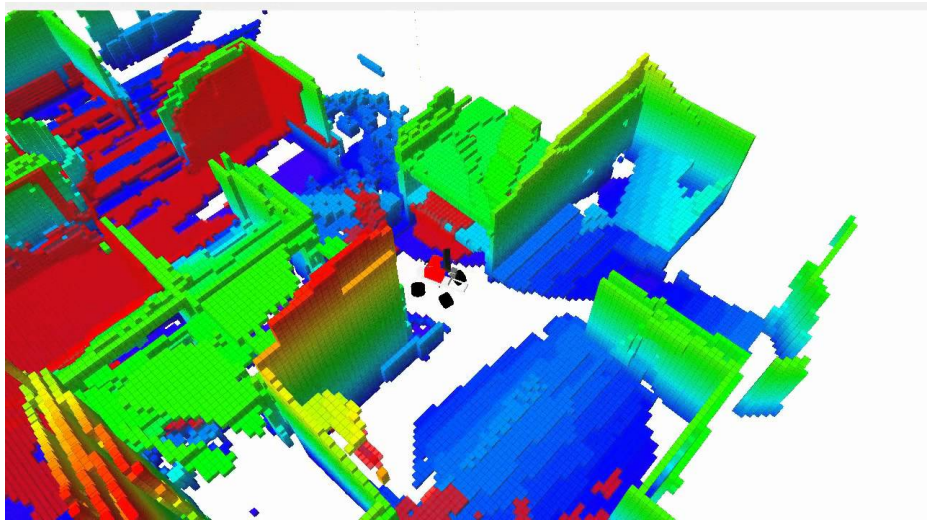
Scaling Event Detection

Many types of events must be handled:

Control Events → Optimize for expiration

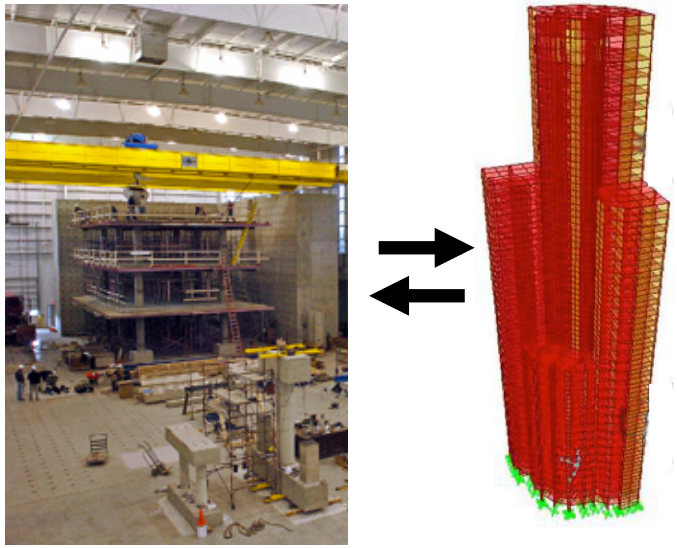
Error Checking → Optimize for non-expiration

Noisy localization



http://ihc2015.info/skin/octomap.akp

Dense data



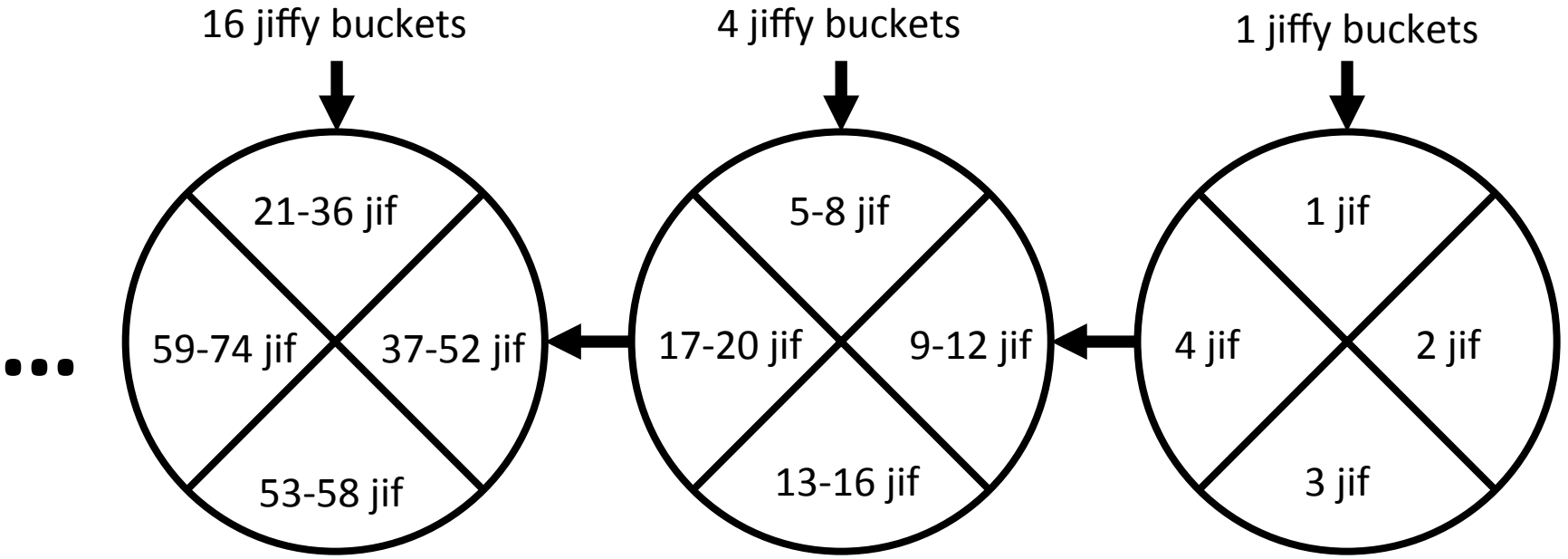
E.g. RT-SfM, SLAM, point clouds, etc.

Optimizing for Non-Expiration

Same problem occurs in network timeouts

- "Millions of network timers" problem
- E.g. per-packet timeouts under TCP
- Adding and removing is common, expiration is not

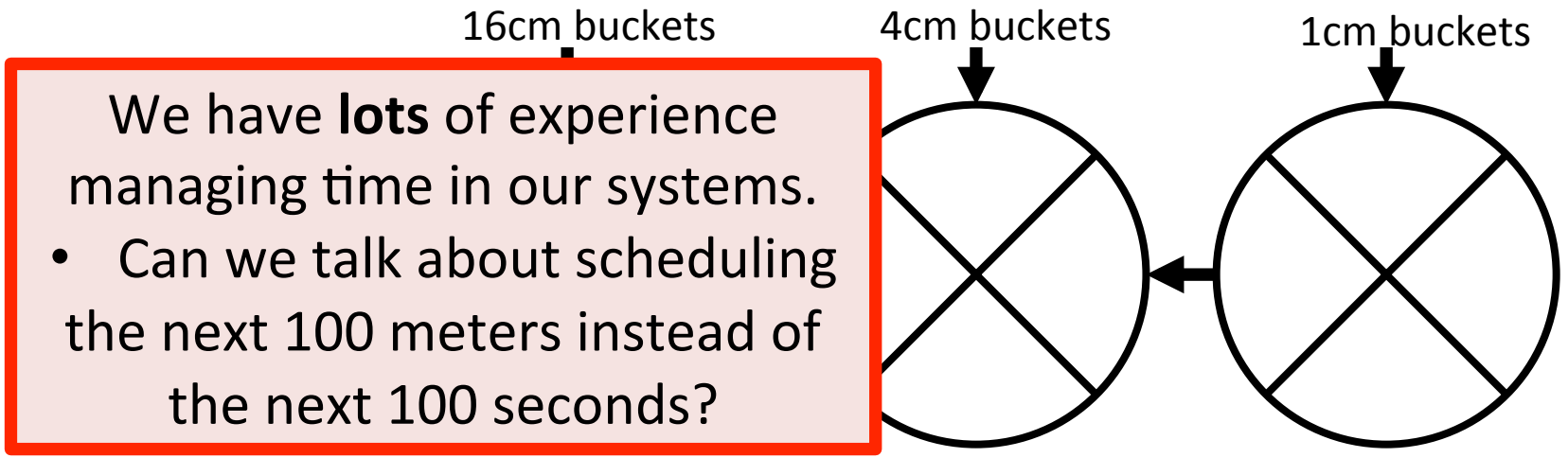
Solution: Timer wheel



Time and Change-Over-Time Duality?

Apply the timer wheel concept to distance?

- Can this physical quantity be treated like time?
  - Directionality, smoothness, and observer uniformity
- Can time be treated like a physical quantity?
  - distance=rate×time goes both ways…



We have **lots** of experience managing time in our systems.

- Can we talk about scheduling the next 100 meters instead of the next 100 seconds?

Dynamics-Based Synchronization

Scalability for CPS:

- Increasingly compute-heavy algorithms
- Increasingly multi-core
- Increasingly parallel and distributed

Common problem: many readers, few writers.

→ Optimize the reader path

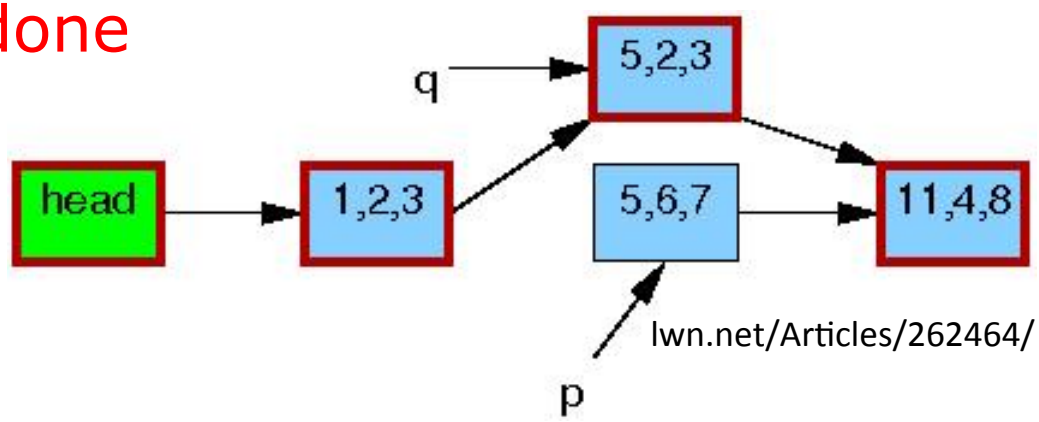
E.g.: Sensor data

Core data structures

Optimizing the Reader Path

Existing solution: Read-Copy-Update (RCU)

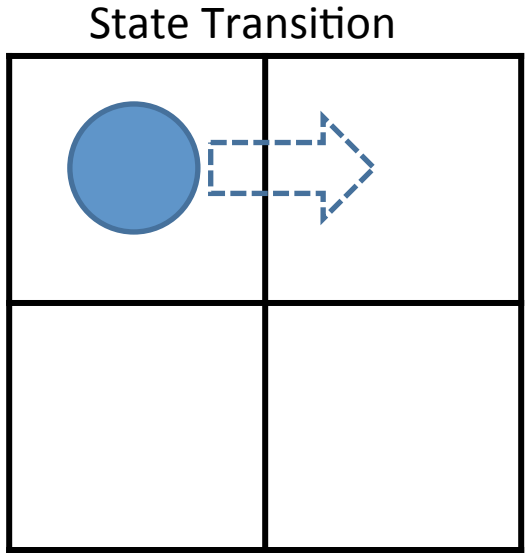
- Writers atomically update data structures
- Reader overhead minimal
- Writer must clean up once it verifies all readers are done



- RCU is simpler in non-preemptive kernels
- Preemption leads to complexities

Dynamics vs. Explicit Ownership?

Dynamics could tell us when data is not needed and an RCU writer can clean up.



- Existing RCU verification can run into the 10's of milliseconds
- Many CPS operate at millisecond-level
- Dynamics lets us make assertions about the global state of a system
- Reclaim simplicity of non-preemptive implementation?

Conclusion?

1) Can scalable OS mechanisms lead to scalable CPS mechanisms?
   - The lens of adapting mechanisms calls out where assumptions differ.

2) Can CPS dynamics lead to cleaner solutions in traditional OS space?
   - "Physics agnostic" systems give a lower bound on CPS performance, adding physical information to a system increases the techniques we might have available to us.
   - Can we surprise the community with a mechanism that works better in a robot or self-driving car than on a desktop PC?
   - Opportunities to send techniques from CPS back up to mainline computing? (E.g. CONFIG_PREEMPT latency)