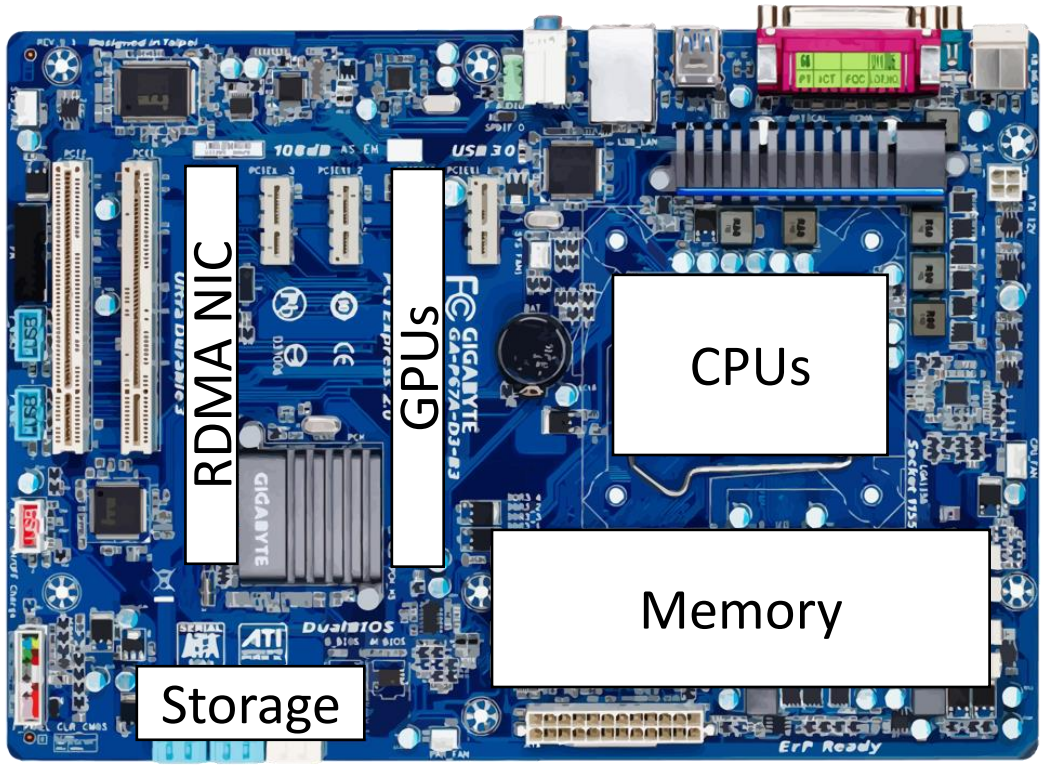
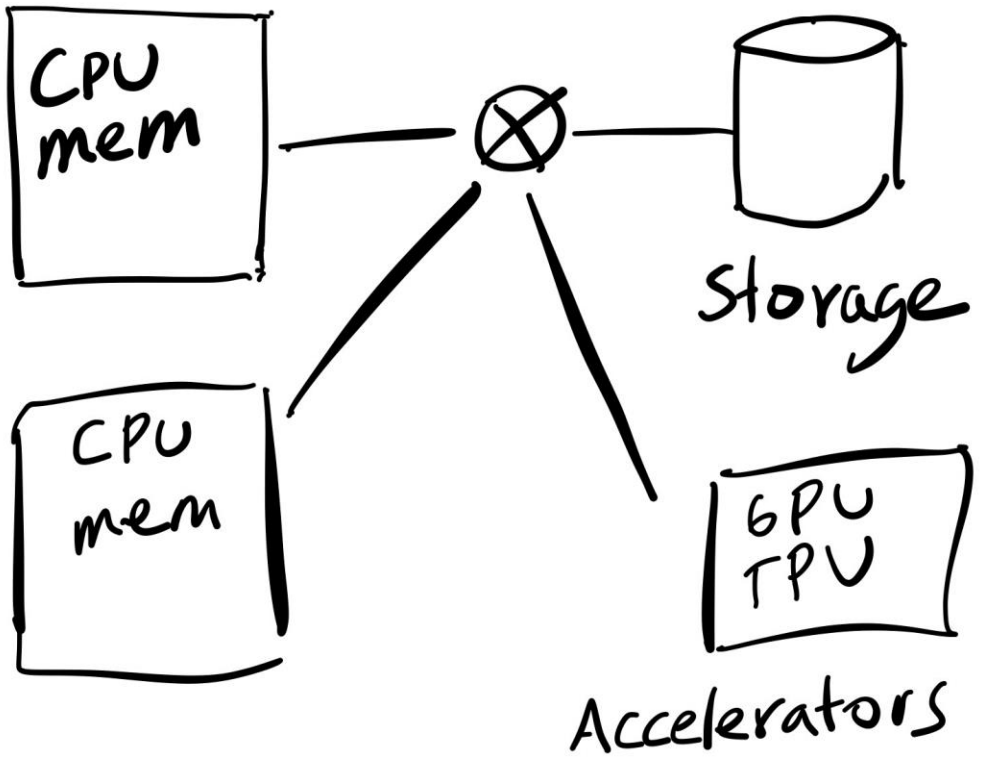
Disaggregation and the Application

Sebastian Angel Mihir Nanavati Siddhartha Sen

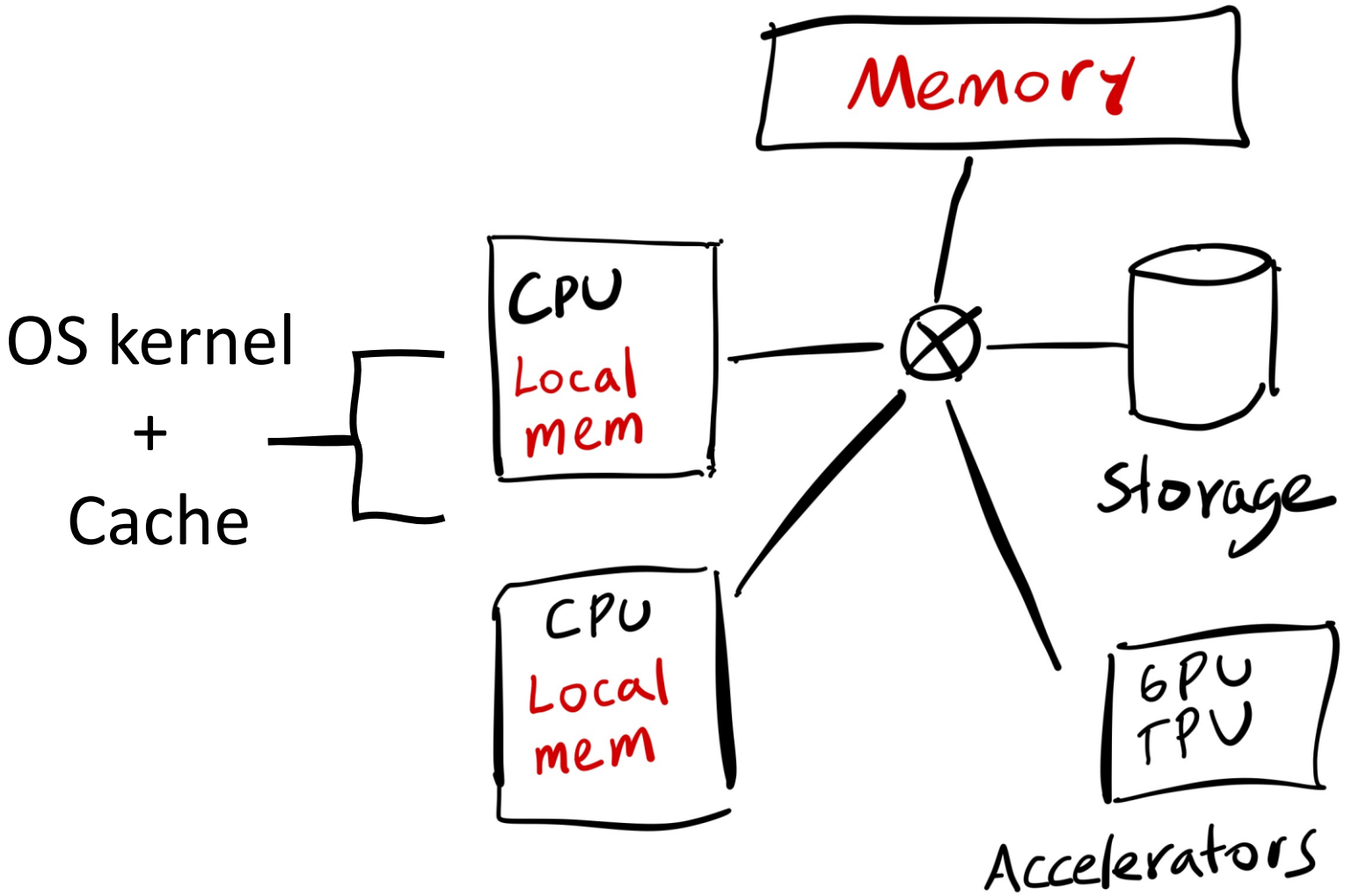
Traditional data center racks



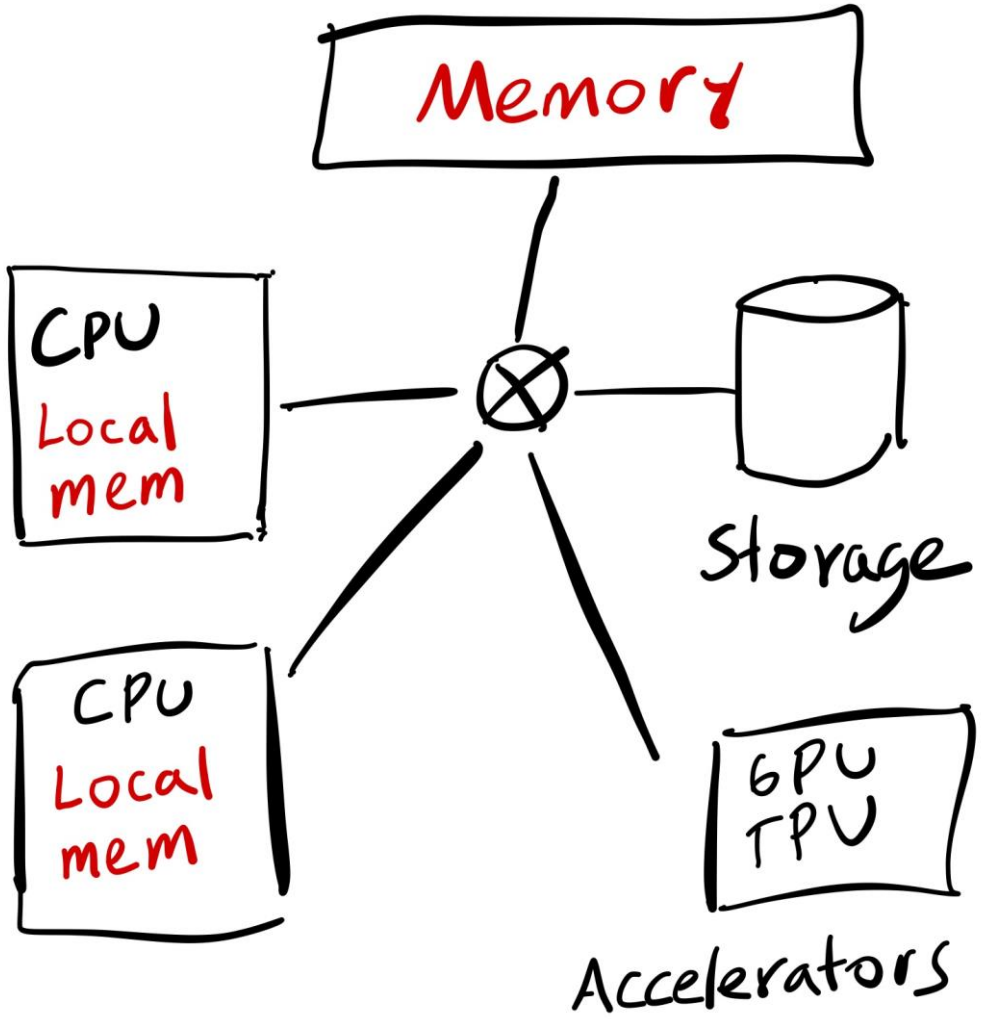
Prior and current disaggregation efforts



Towards DDCs



Why? Many benefits for operators



1) Independence
   - Evolve independently
   - Scale independently
   - Fail separately

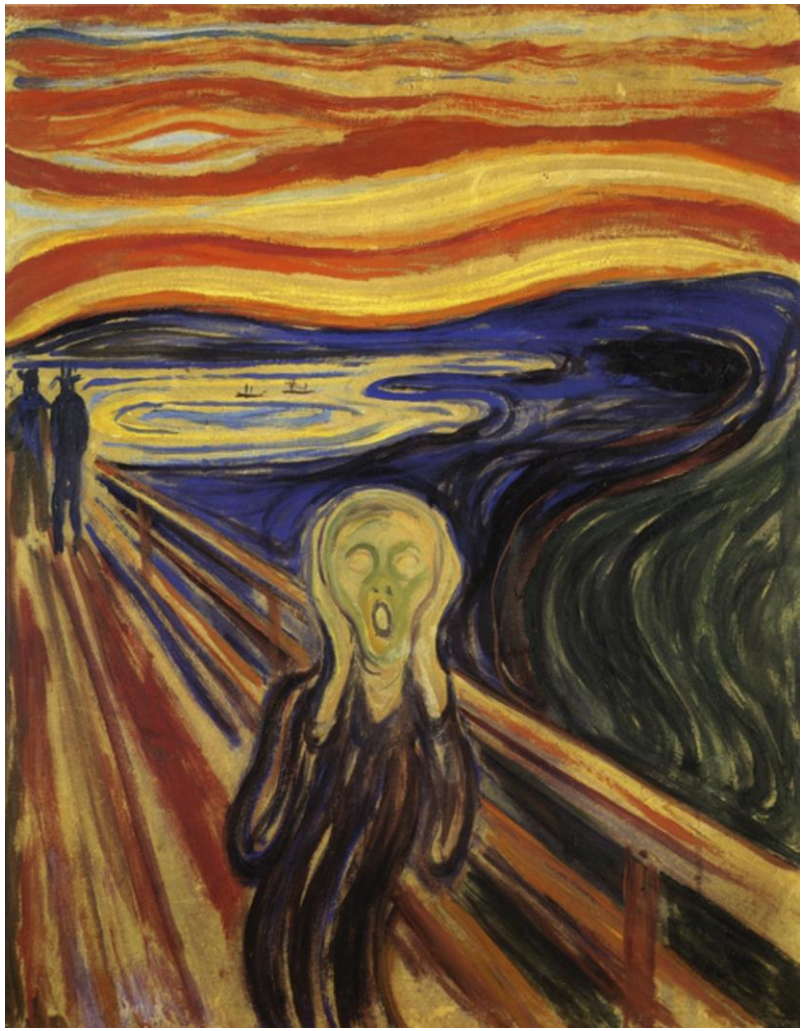
2) Flexible provisioning

3) Less waste

Can you run regular applications on DDCs?

Yes! OSes such as LegoOS [SOSP ‘18]
provide a transparent POSIX API

Should you run regular applications on DDCs?



Understanding the Effect of Data Center Resource Disaggregation on Production DBMSs

Qizhen Zhang, Yifan Cai, Xinyi Chen, Sebastian Angel
Ang Chen, Vincent Liu, Boon Thau Loo

University of Pennsylvania, Shanghai Jiao Tong University, Rice University

ABSTRACT

Resource disaggregation is a new architecture for data centers in which resources like memory and storage are decoupled from the CPU, managed independently, and connected through a high-speed network. Recent work has shown that although disaggregated datacenters (DDCs) provide operational benefits, applications running on DDCs experience degraded performance due to extra network latency between the CPU and their working sets in main memory. DBMSs are an interesting case study for DDCs for two main rea-sons: (1) DBMSs normally process data-intensive workloads and rrequire data movement between n the CPU and their working sets in main memory.DBMSs are an interesting case study for DDCs for two main rea-sons: (1) DBMSs normally process data-intensive workloads andrequire data movement between different resource components; and(2) disaggregation drastically changes the assumption that DBMSscan rely on their own internal resource management.

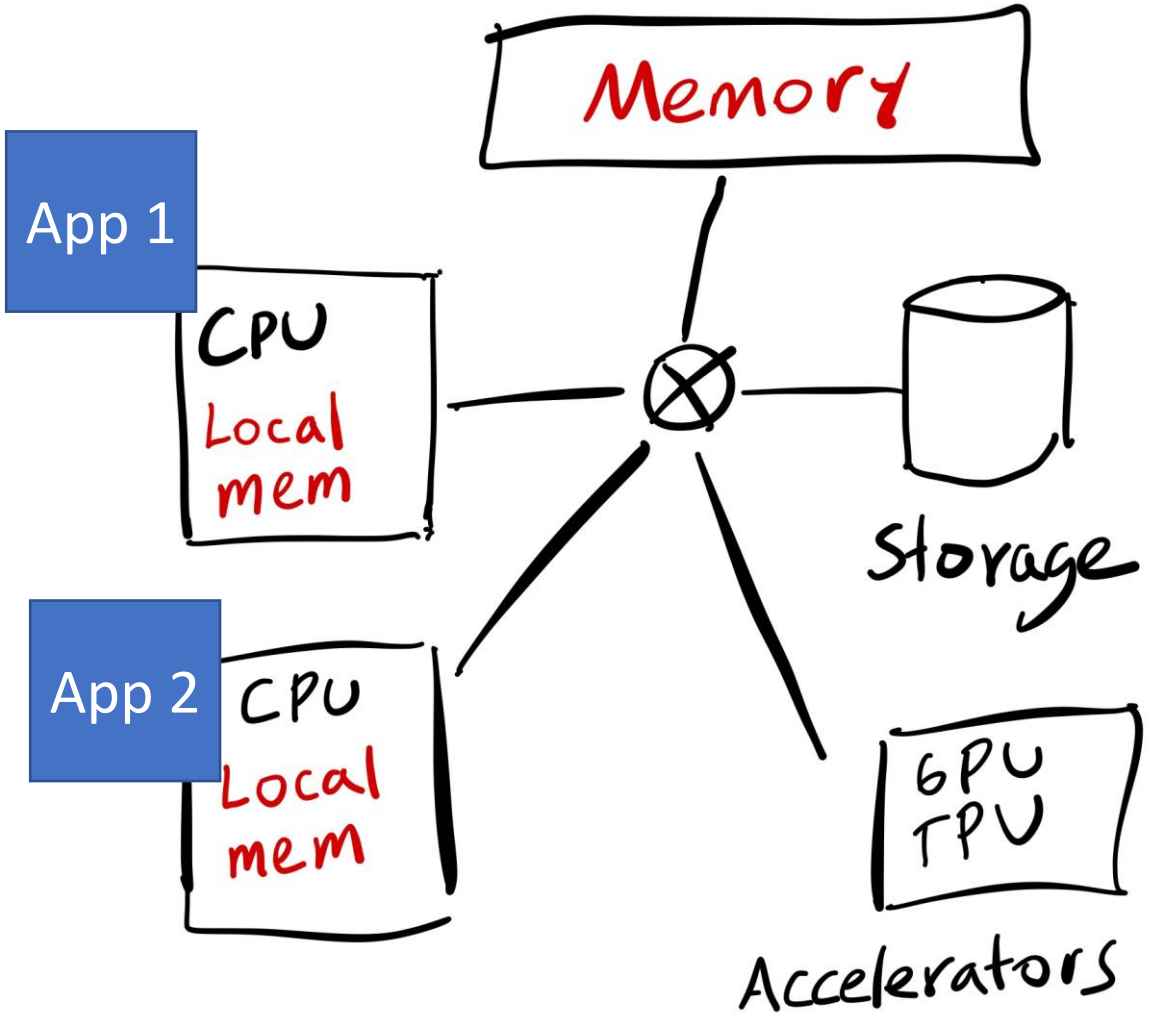
others for storage. To complete a single task, a processing node will need to continually "page" memory from remote nodes into and out of its small on-board working set, write chunks to remote disks, or farm out tasks to remote CPUs or GPUs.

Disaggregating resources in this way provides substantial benefits to data center operators. It allows them to upgrade and expand each resource independently, e.g., if a new processor technology becomes available or if the workload changes require additional CPUs. It also allows them to prevent fragmentation and over-provisioning, e.g., if a customer requests an unusual balance between CPU cores, , RAM, and GPUs that does not fit neatly into an existing machine. Finally, to users, disaggregation creates the illusion of a near-infinite pool of any resource for any program. Disaggregation has fundamental implications on the performance of data-intensive applications, not all of which are positive. For example, our recent work [8, 38] highlights the

Summary: terrible performance

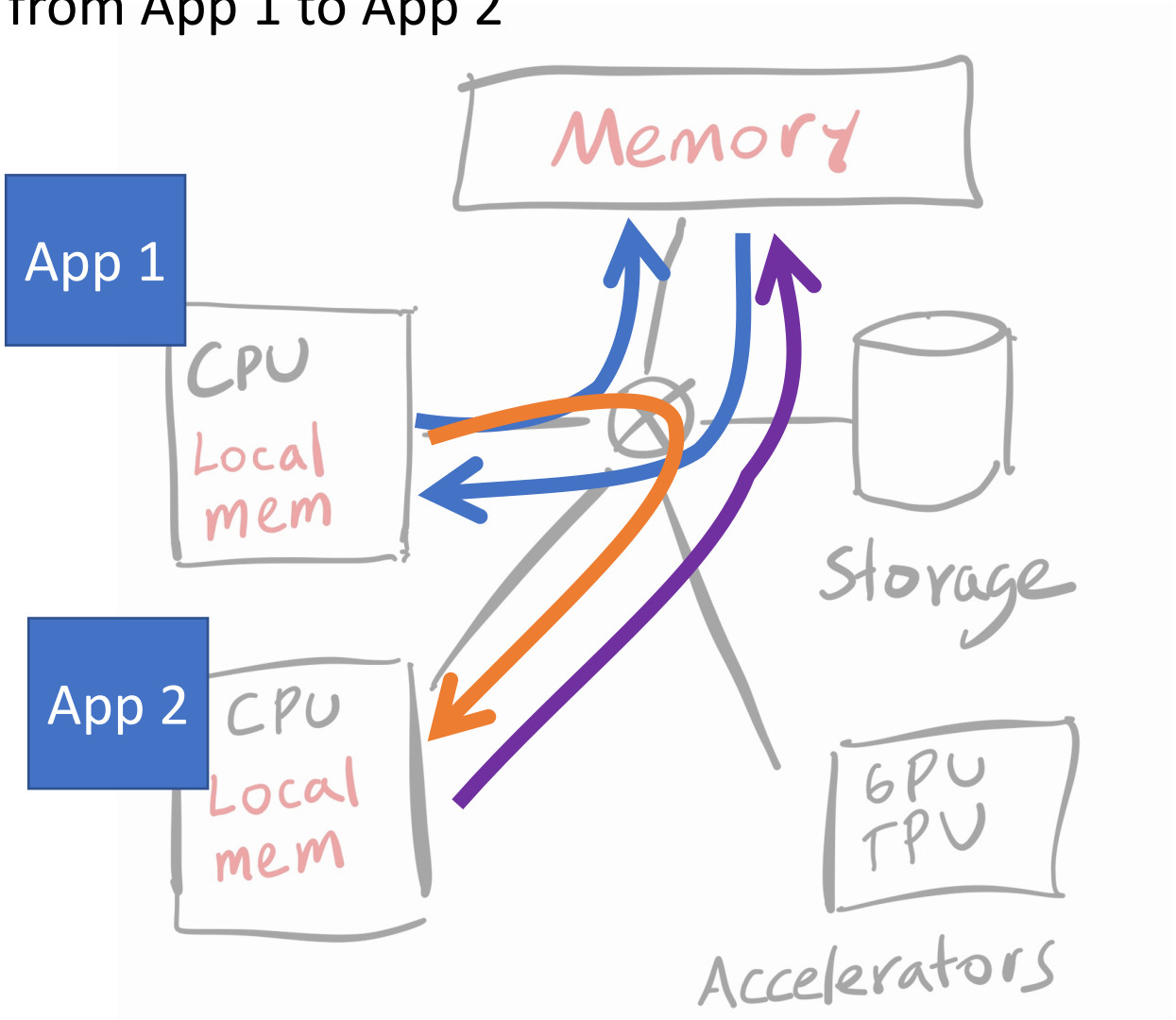
Key issue: Too much data movement

Goal: send data from App 1 to App 2



Key issue: Too much data movement

Goal: send data from App 1 to App 2



Our position:

OSes should expose the disaggregated nature of DDCs to applications and let them exploit it for their benefit

In the rest of this talk

- What abstractions should DDC OSes expose to applications?
- Which applications can benefit from these abstractions?

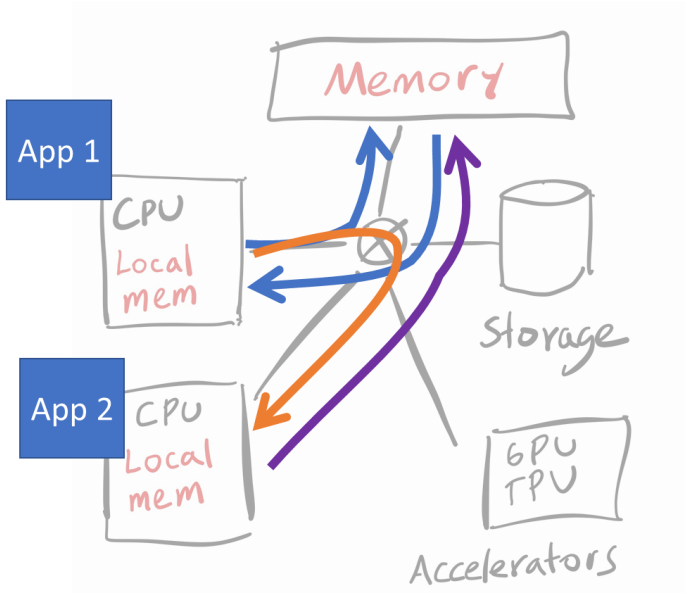
OSes can expose:

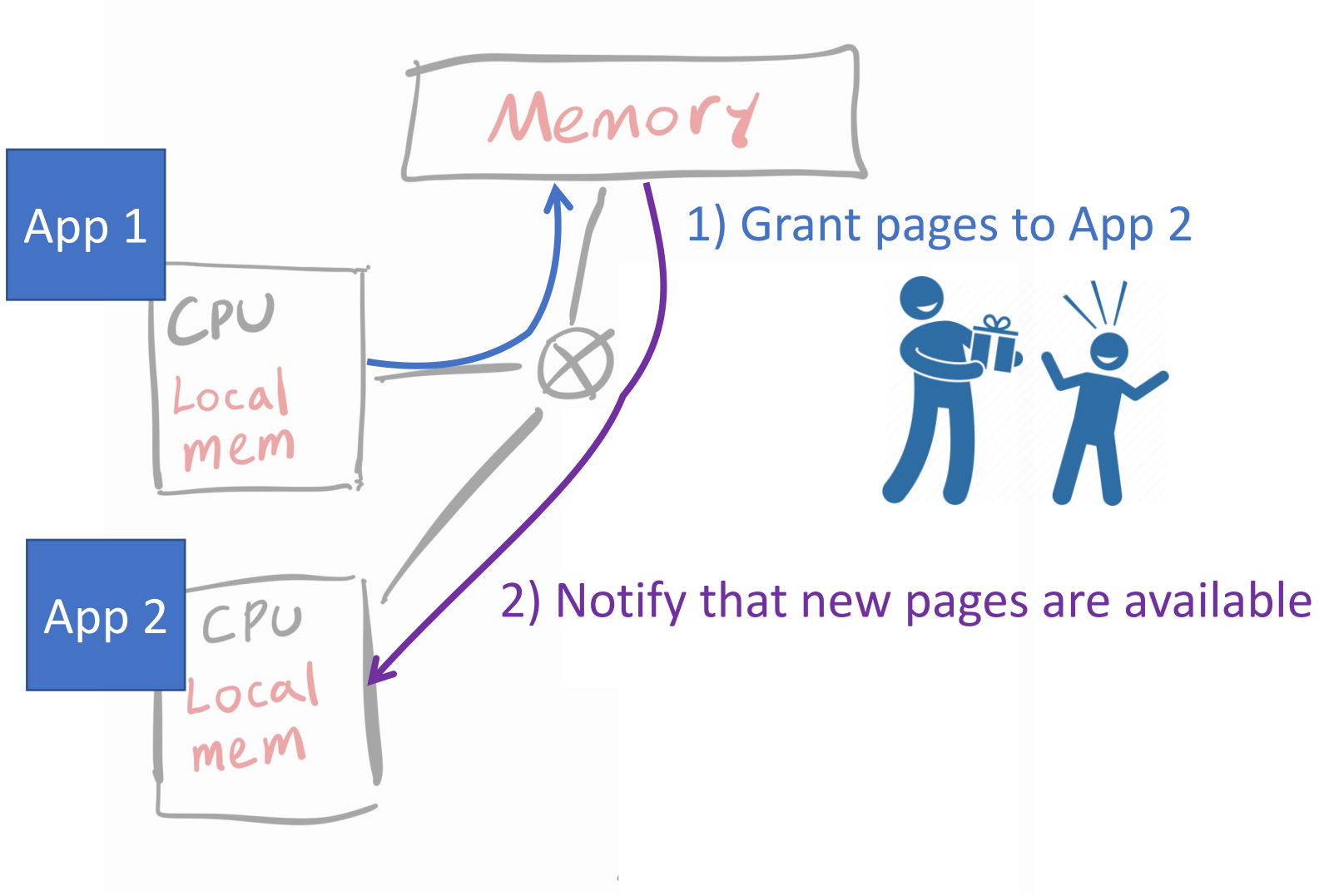
- That processes access the same memory nodes
- Failure independence
- Memory nodes might have a CPU/FPGA
  - Useful for near-data processing / computation offloading

We propose three new OS abstractions

- Memory grant
- Memory steal
- Failure informers / Spies

Memory grant



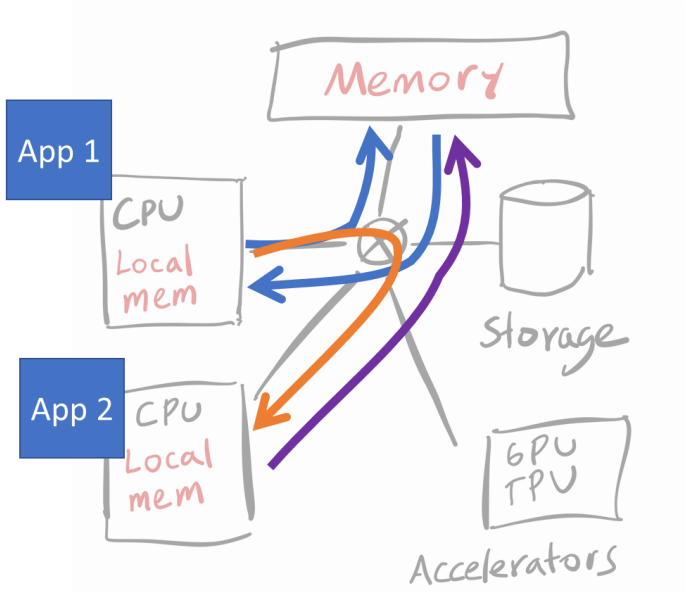


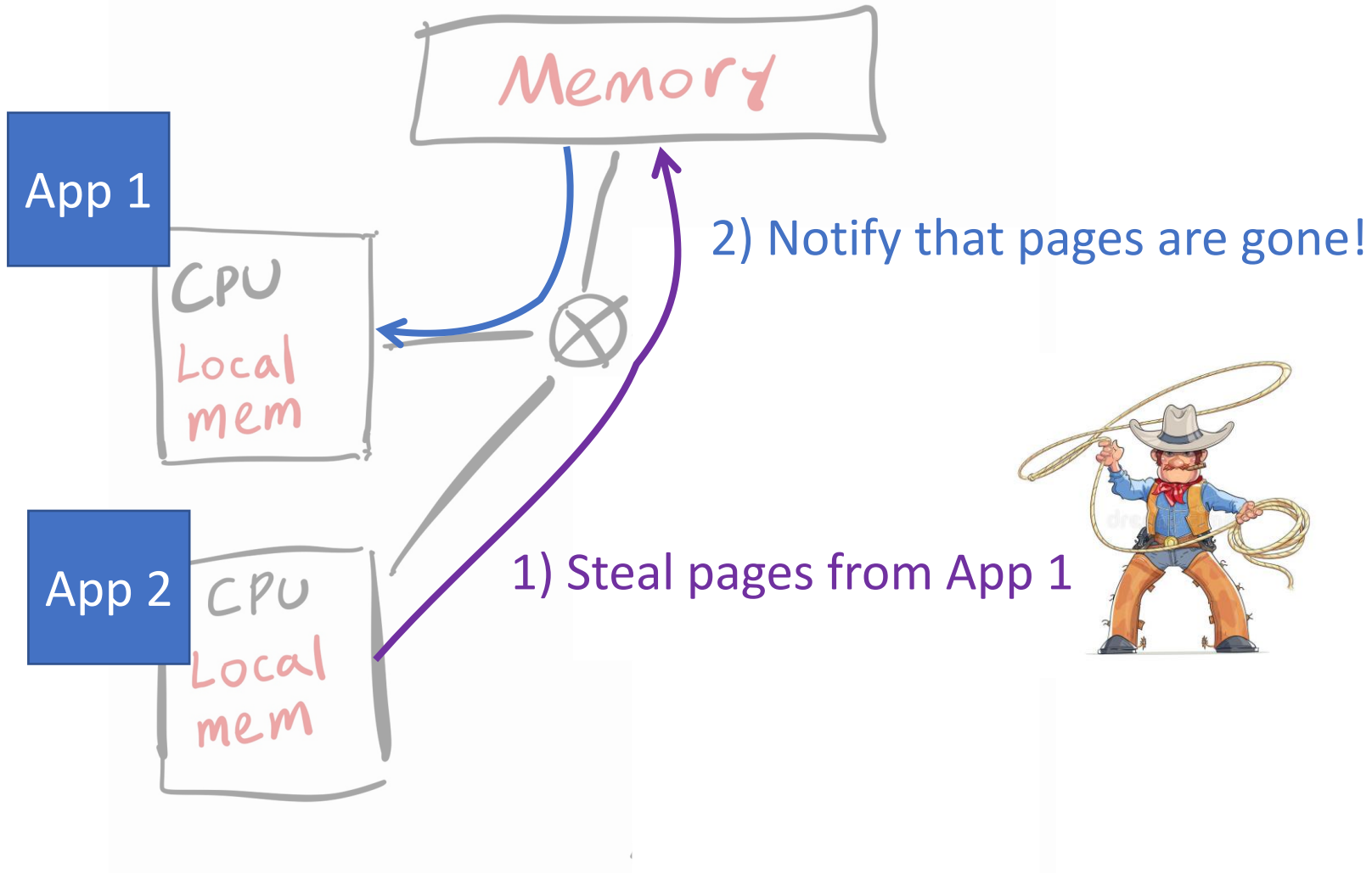
Properties of Grant

- Grant has **move** semantics
  - Grantor loses access to the memory
  - Similar to `vmsplice` with "`GIFT`" flag in Linux

- Virtual memory addresses remain the same
  - To preserve correctness of internal references
  - **Problem**: what if grantee already used those addresses?

Memory steal

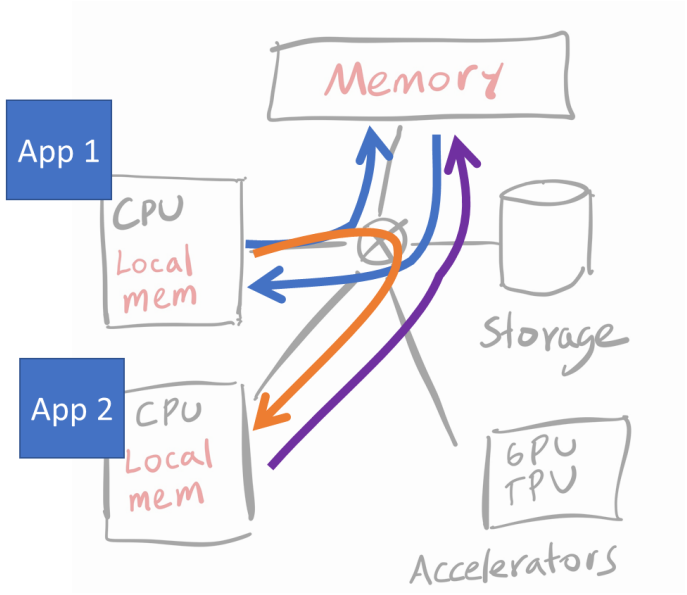


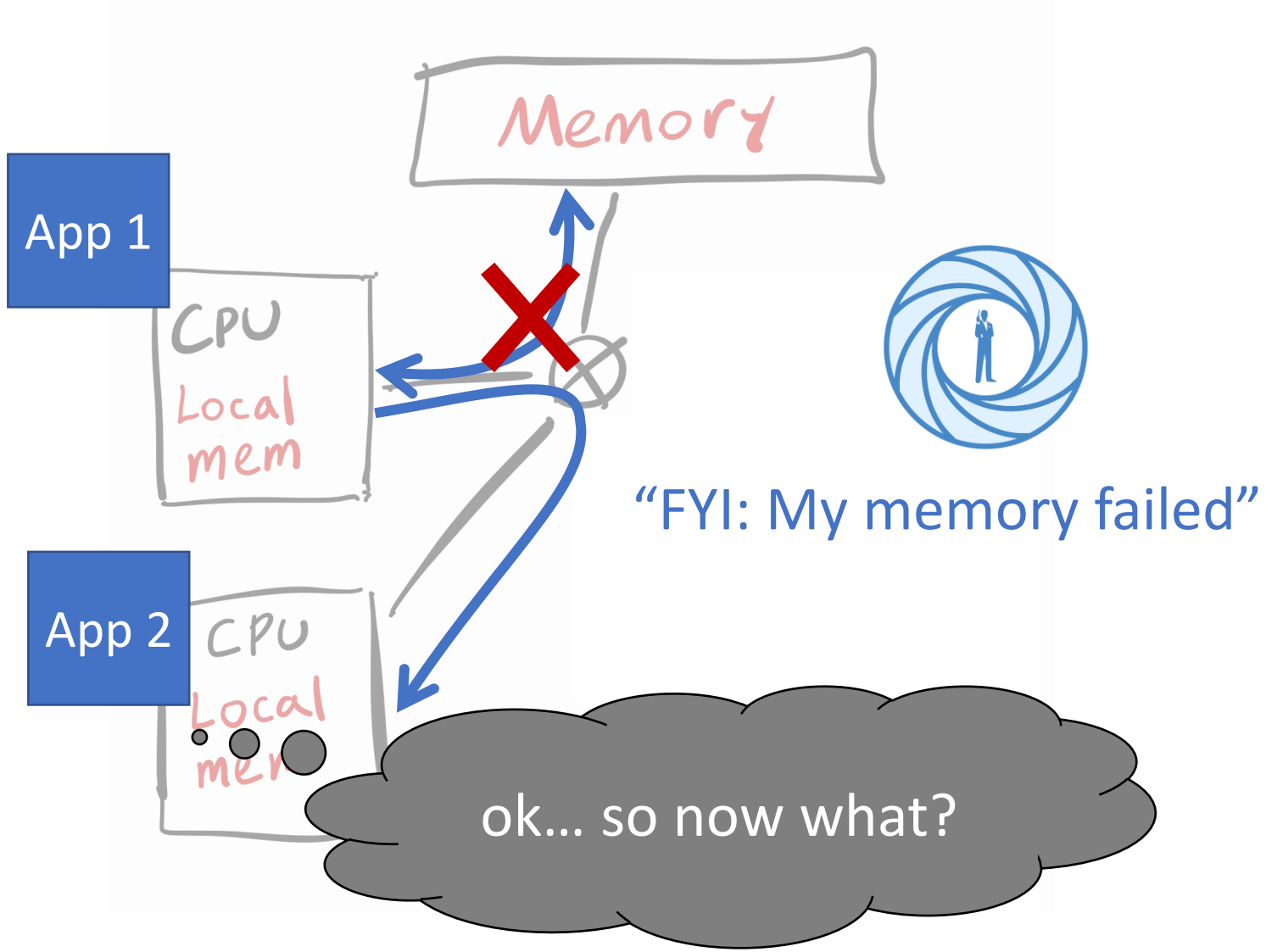


Properties of Steal

- Same semantics as **Grant**
  - But is involuntary: Can happen at any time
- Meant to be used by different instances of the **same app**
  - Can coordinate through the network / use capabilities
  - Incorrect steal = bug
- Must ensure stolen **memory is consistent**
  - Can model with crash consistency

Failure informers / Spies





In the rest of this talk

- What abstractions should DDC OSes expose to applications?
- Which applications can benefit from these abstractions?

Some applications

- Dataflow applications could
  - Use **Grant** to pass data around
  - Use **Steal** to deal with stragglers

- New memcached instances can **Steal** part of object space (scale out)

- Paxos can use **failure informers** for quicker reconfigurations
  - Memory dies → Paxos replica informs others and then kills itself
  - CPU dies → New replica takes over the dead CPU's memory and keeps going

Summary

Running existing applications on DDC is not advisable

There is potential in modifying apps to exploit the nature of DDCs

OSes should expose more information and control to applications

Grant

Steal

Spy