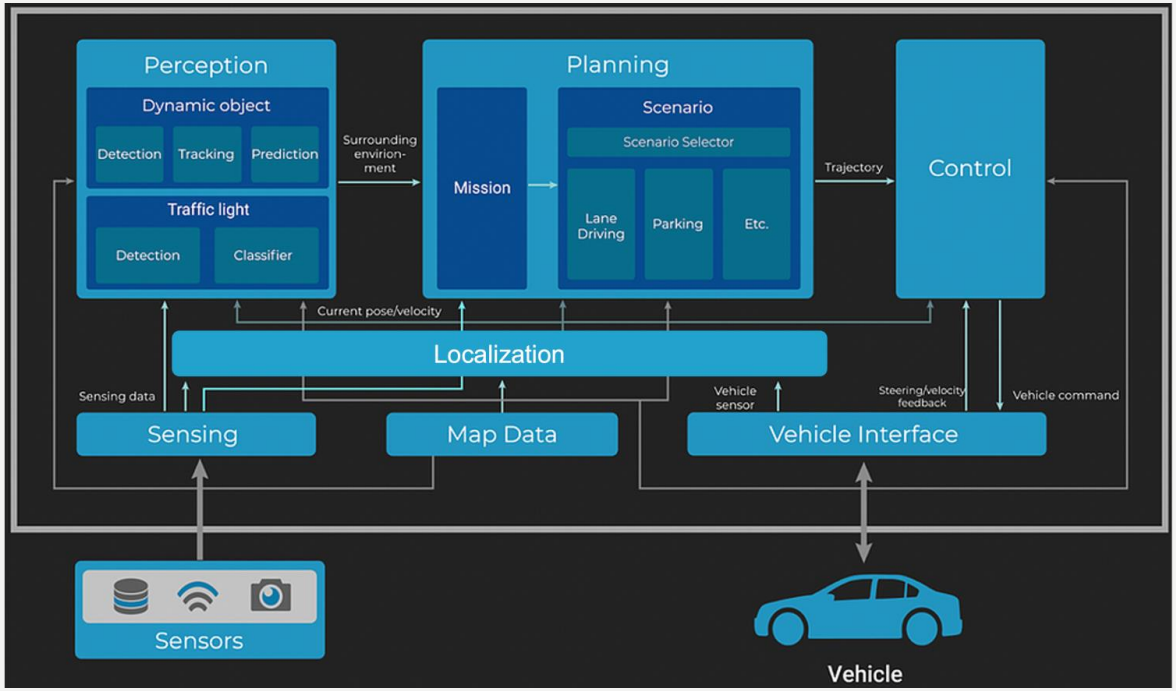
HARDWARE ACCELERATION WITH ZERO-COPY MEMORY MANAGEMENT FOR HETEROGENEOUS COMPUTING

OREN BELL
CHRIS GILL
XUAN ZHANG

CPS-1739643

DESIGN OF A SELF-DRIVING CAR



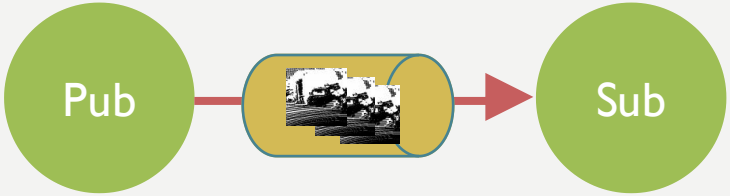
THE AUTOWARE FOUNDATION

HAZCAT

- Communication middleware
- Zero-copy
- Tested with ROS2 applications
  - ROS2 is framework for developing robotics
- Support for pub/sub application

INTRO TO PUBLISH/SUBSCRIBE (PUB/SUB)

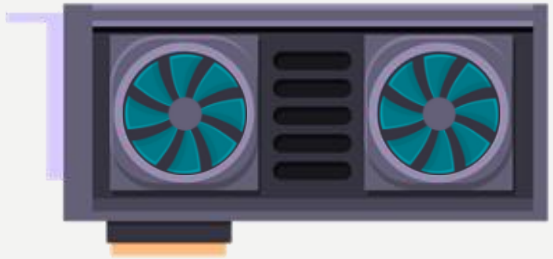
- Topic represents portion of system state
  - Frame from front camera
  - Planned trajectory
  - Etc
- Publishers send data
- Subscribers receive data
- Data called "messages"
- Topic modelled as message queue

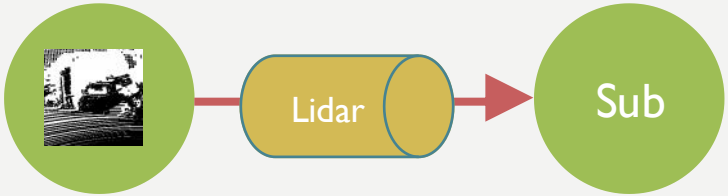


USING THE GPU

- Copying in and out of GPU is bad
  - Time consuming
  - Synchronization can affect unrelated tasks using it
- **Goal: keep data in GPU**

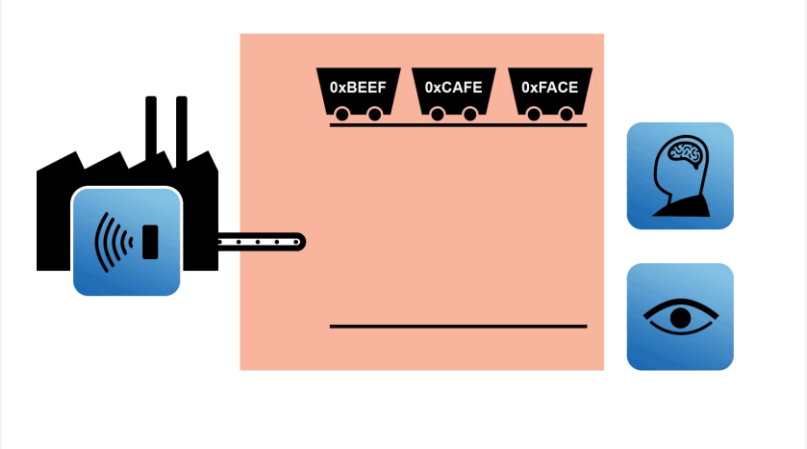
- Related problem: copying data into topic
  - Existing solution: Zero-copy





RELATED WORK: ICEORYX©

- Inter-process zero copy
- Central pool of shared memory buffers
  - Various sizes
  - Configured and pre-allocated prior to runtime
- Borrow/return semantics
  - Publishers surrender ownership after publication
  - Messages marked immutable
  - Prevents race conditions

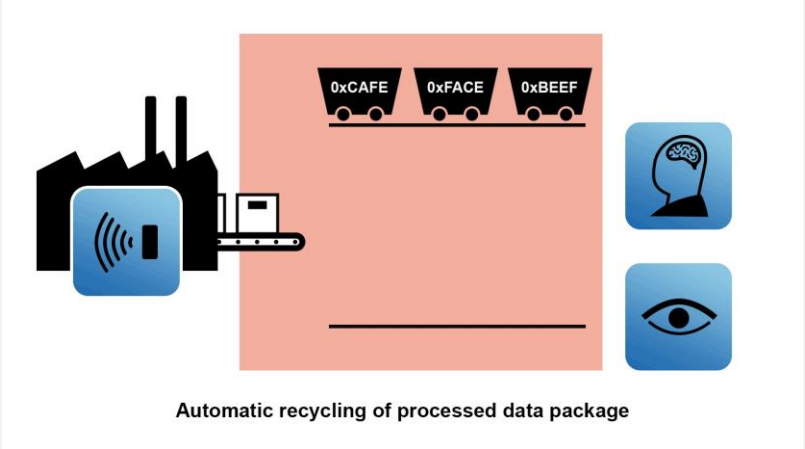


RELATED WORK: ICEORYX©

ECLIPSE FOUNDATION
BOSCH
Apex.AI

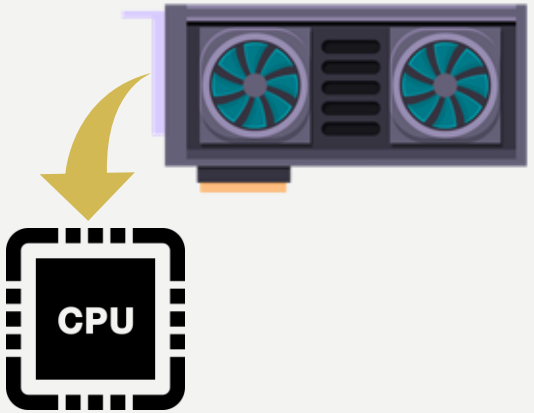
Drawbacks

- Centralized bottleneck
- Must be configured by skilled developer
- Shared memory only accessible from CPU



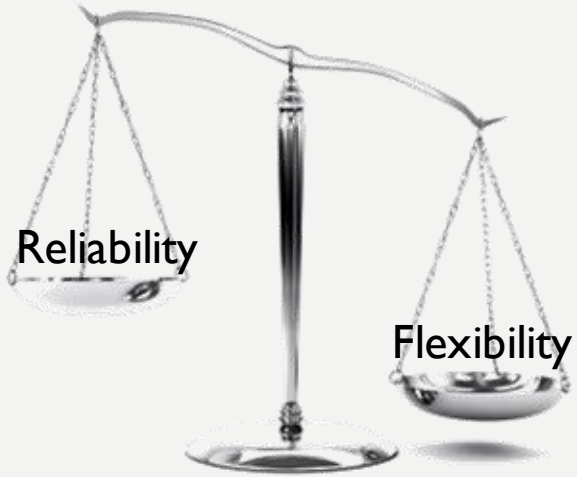
ZERO-COPY: OUR GOALS

- Support for heterogenous memory
  - Zero copy within a memory domain



ZERO-COPY: OUR GOALS

- Support for heterogenous memory
  - Zero copy within a memory domain
- Custom allocation strategies
  - Best-effort dynamic allocation
  - Reliable static pre-allocation

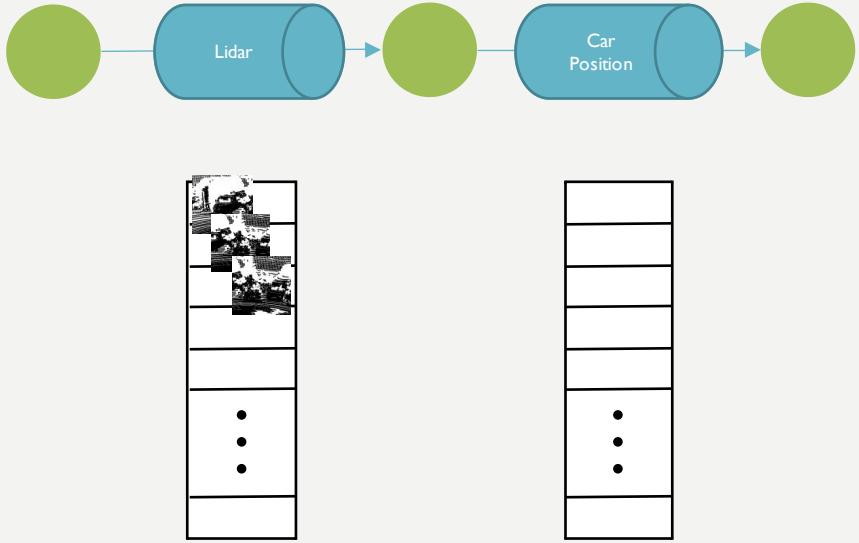


ZERO-COPY: OUR GOALS

- Support for heterogenous memory
  - Zero copy within a memory domain
- Custom allocation strategies
  - Best-effort dynamic allocation
  - Reliable static pre-allocation
- Prevent race conditions

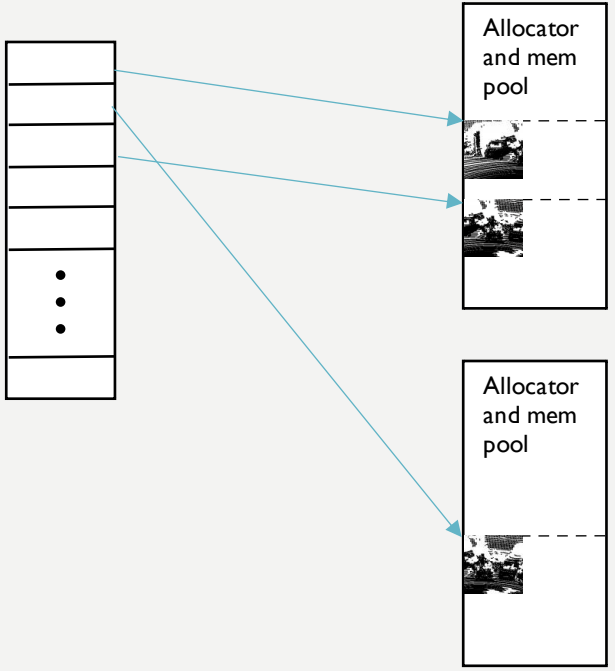
ZERO-COPY: DECENTRALIZED MEMORY MANAGEMENT

- One queue per topic
  - Resides in shared memory
- Topic name for lookup
- Resized during publisher/subscriber registration
  - Subscribers inform size of message queue
  - Initialization cost



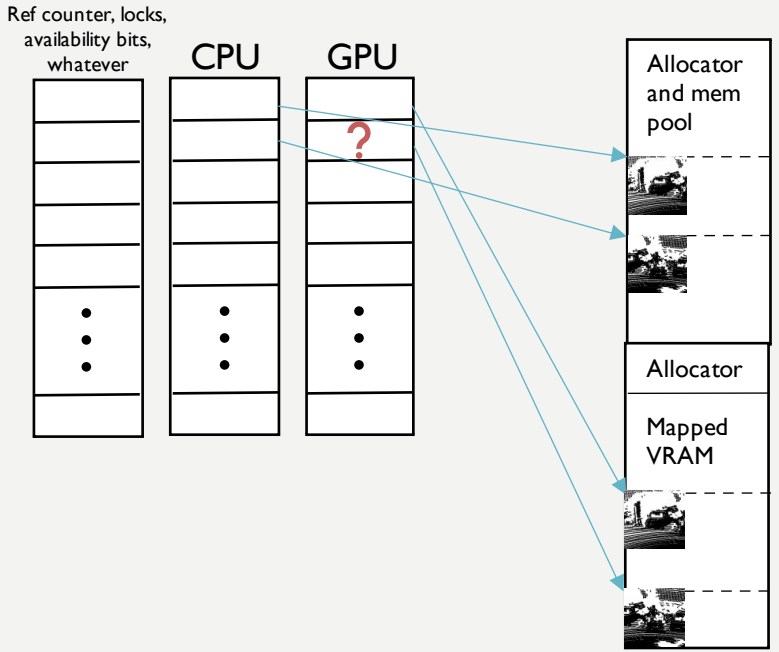
ZERO-COPY: CUSTOM ALLOCATION STRATEGIES

- Queues store tokens
  - Tokens are allocator ID + offset
  - Footprint of buffers very small
- Allocators exist in shared memory
  - Accessible from multiple processes



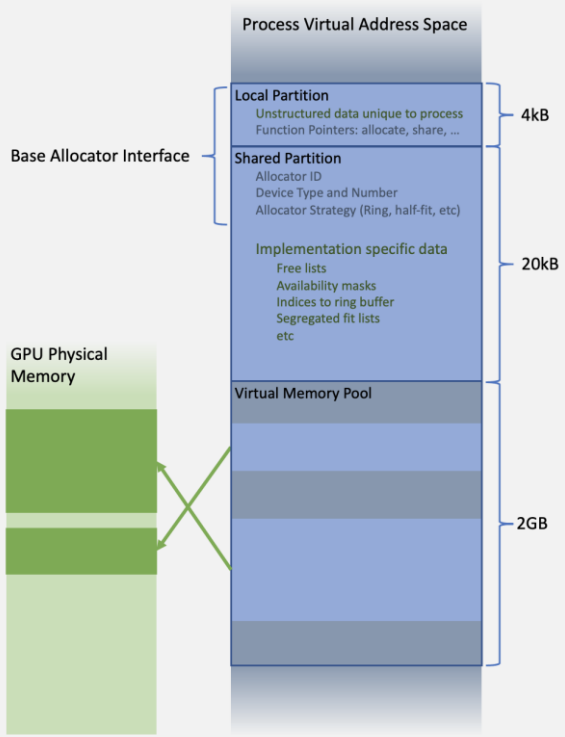
ZERO-COPY: HETEROGENOUS MEMORY SUPPORT

- Extra queue per memory domain
  - Extra queues store extra message copies
  - Additional queue with metadata
- Allocators support only one domain
- Publishers/subscribers specify allocator preference
- Copying between domains performed as needed



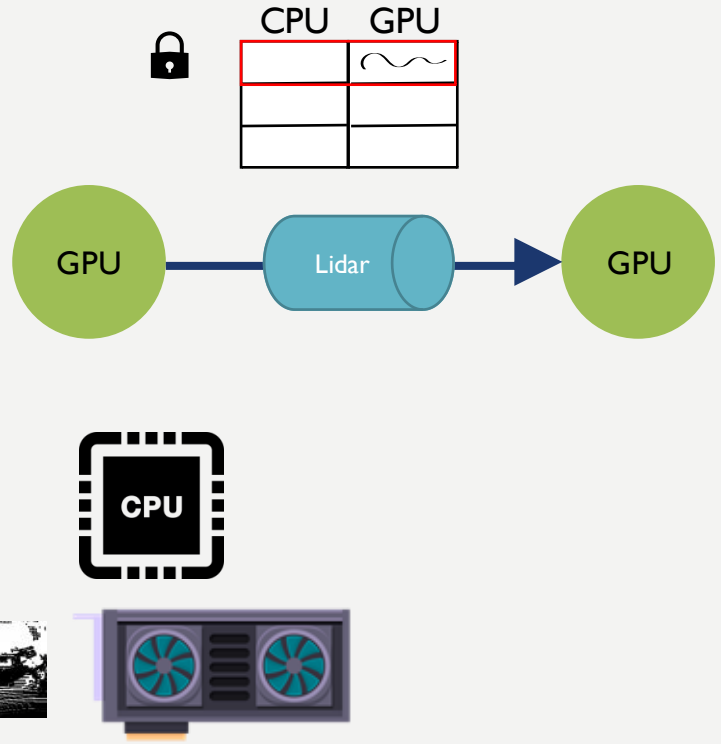
ALLOCATOR DESIGN

- Local Partition
  - Pointers to function implementations
- Shared Partition
  - Metadata and overhead
- Memory Pool
  - Virtual, often sparsely mapped
  - Actual data
  - Assumed unreadable by host code



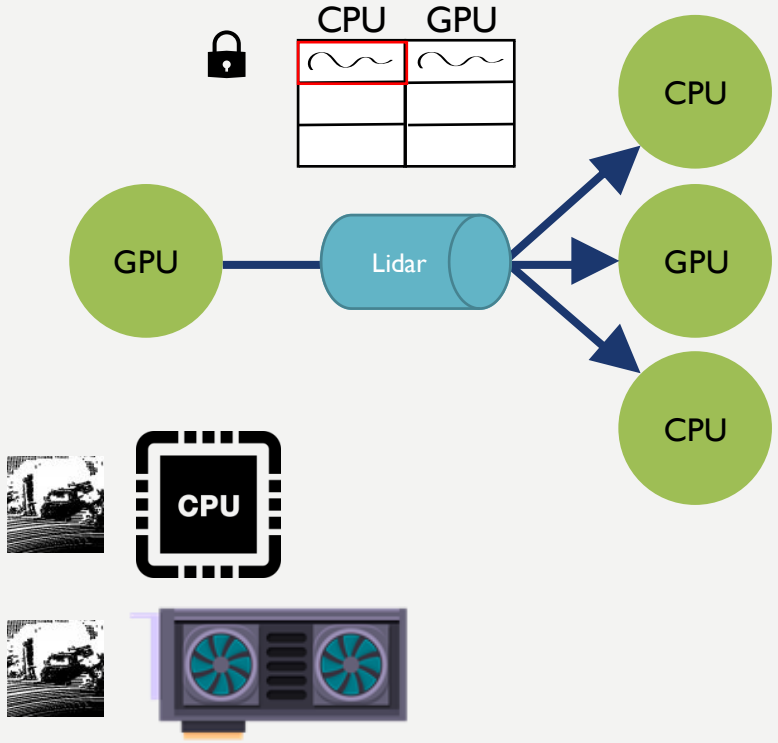
PUBLICATION PROCEDURE

1. Pub creates message with allocator
2. Lock free row in queue
3. Token stored in GPU field
4. Pub frees lock



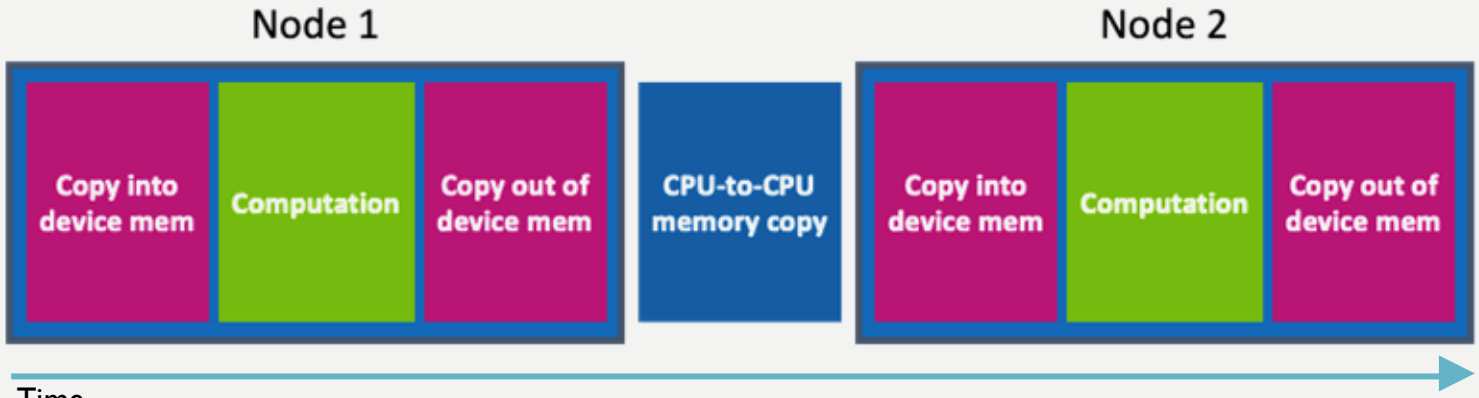
SUBSCRIPTION PROCEDURE

1. If message not in preferred domain
    1. Lock CPU field
    2. Copy of message is made
    3. Store token in CPU field
    4. Subscriber frees lock
2. Read message from token
3. If all subscribers done, deallocate message

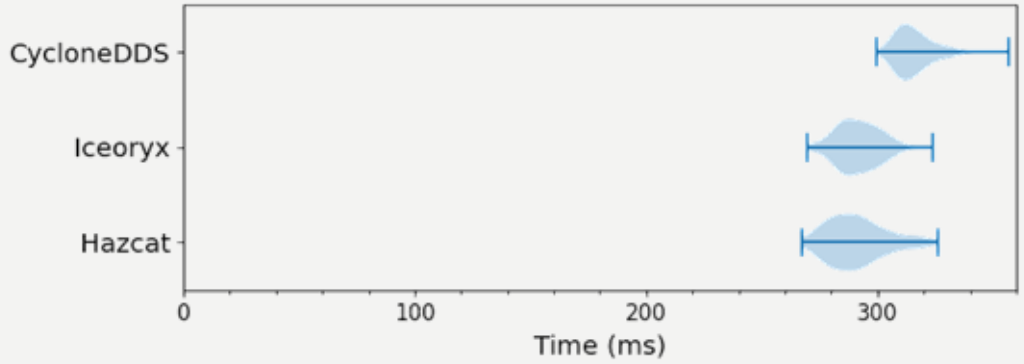


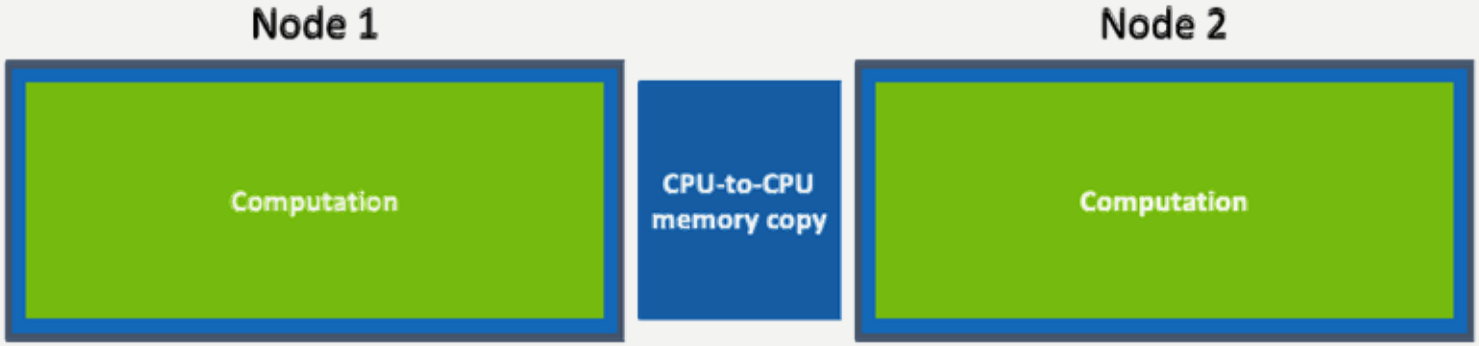
TWO-COMPONENT EXPERIMENT

- Inter-component copy
  - Copy messages between nodes
- Intra-component copy
  - Copy data into hardware accelerated device
- Computation

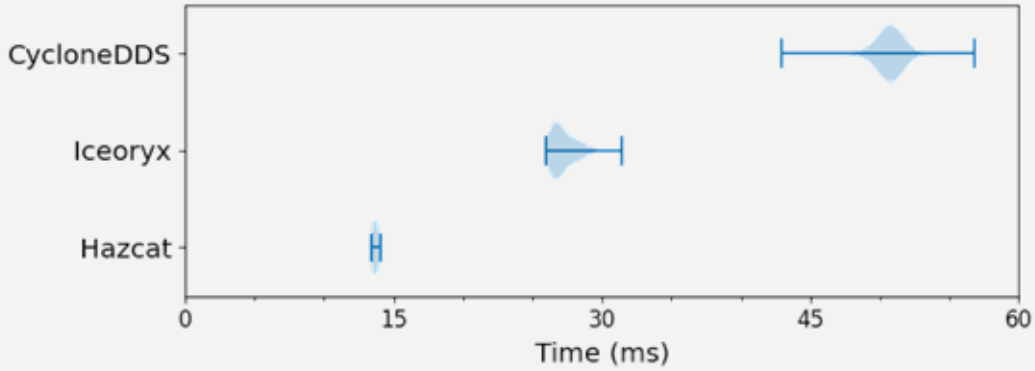


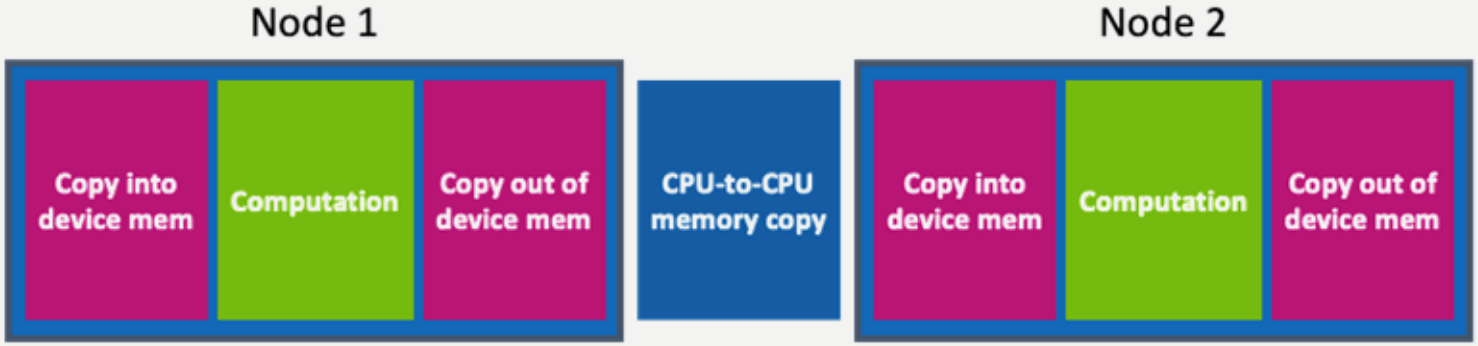
WITHOUT HARDWARE ACCELERATION





WITH HARDWARE ACCELERATION



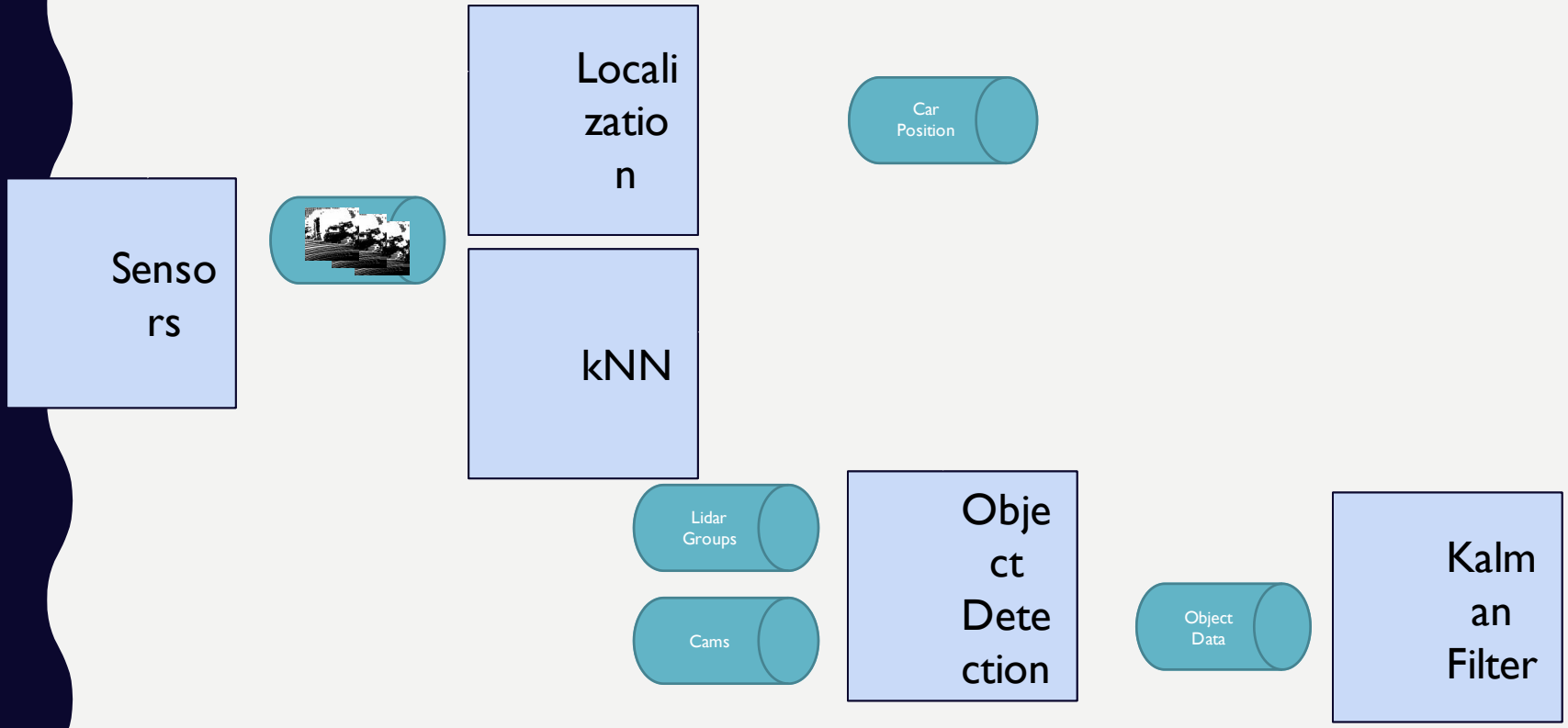


CONCLUSIONS

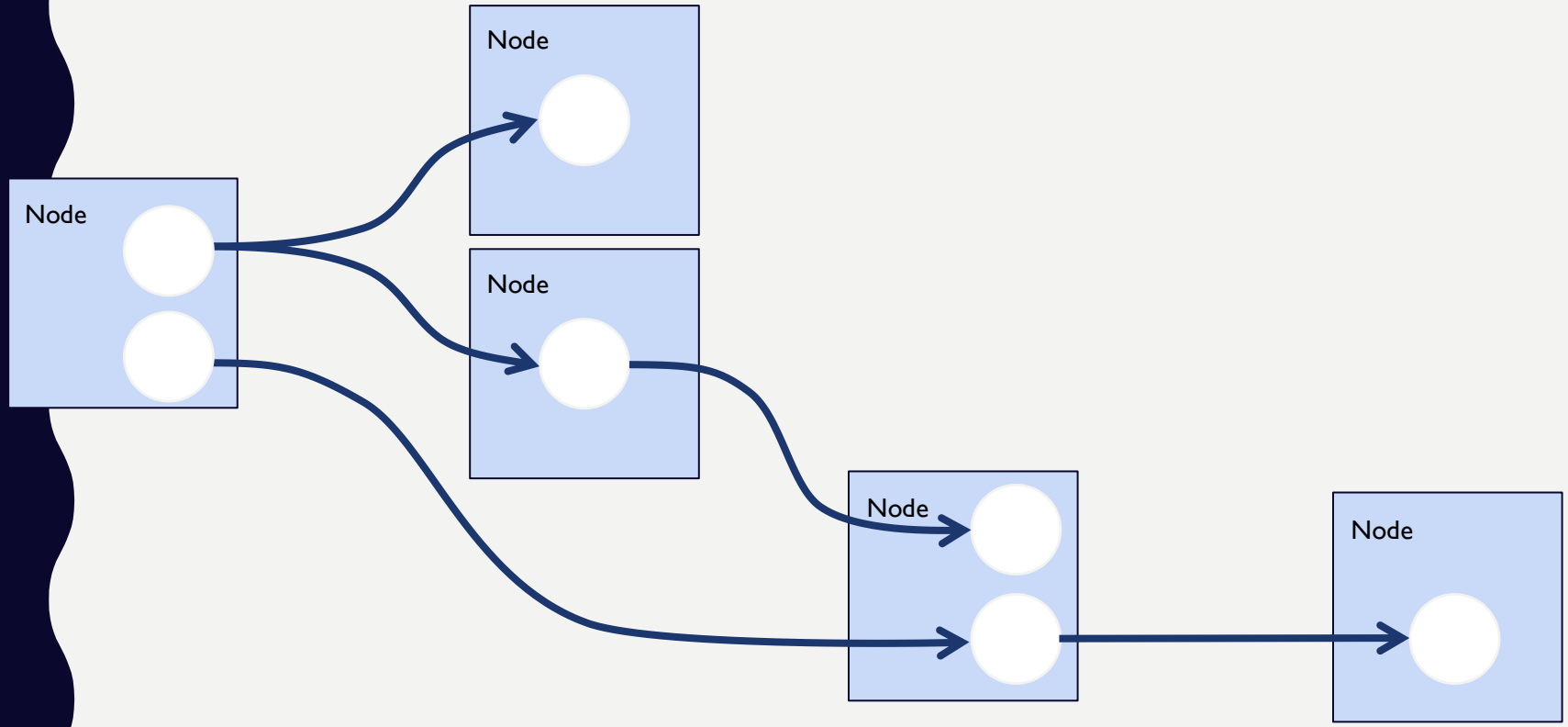
- Zero-Copy System
  - Heterogenous-memory awareness
  - No configuration needed beforehand
    - Created/resized at runtime during initialization
  - Duplicate copies of messages for each memory domain
- Benefits
  - Cap on memory operations
    - For n domains, max of n-1 copy operations per message
  - Reduced opportunity for indeterminism and unwanted synchronization
  - Highly performant, in principle
- GitHub: https://github.com/nightduck/hazcat

ROS2 OVERVIEW

ROS2 Overview: Pub/Sub

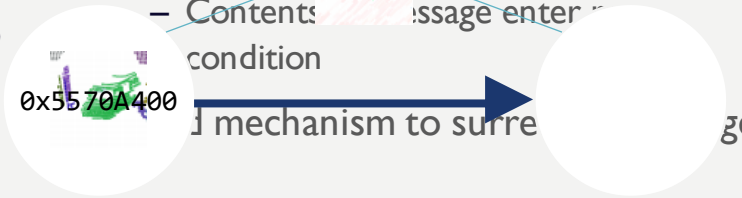


ROS2 OVERVIEW: PROCESSING CHAINS



INTRO TO ZERO-COPY

- What if task keeps running after publishing?
  - Contents ssage enter
    condition



d mechanism to surre ges

ALLOCATORS

- Responsible for allocating/deallocating memory
- Static allocators pre-allocate memory pools

```
template<class T>
struct allocator {
  T* allocate(int size) {
    return malloc(size * sizeof(T));
  }
  void deallocate(T *p) {
    free(p);
  }
}
```

INTRA-PROCESS ZERO COPY FOR GPUS

Template for heterogenous memory allocators (language-agnostic)

Defined by user

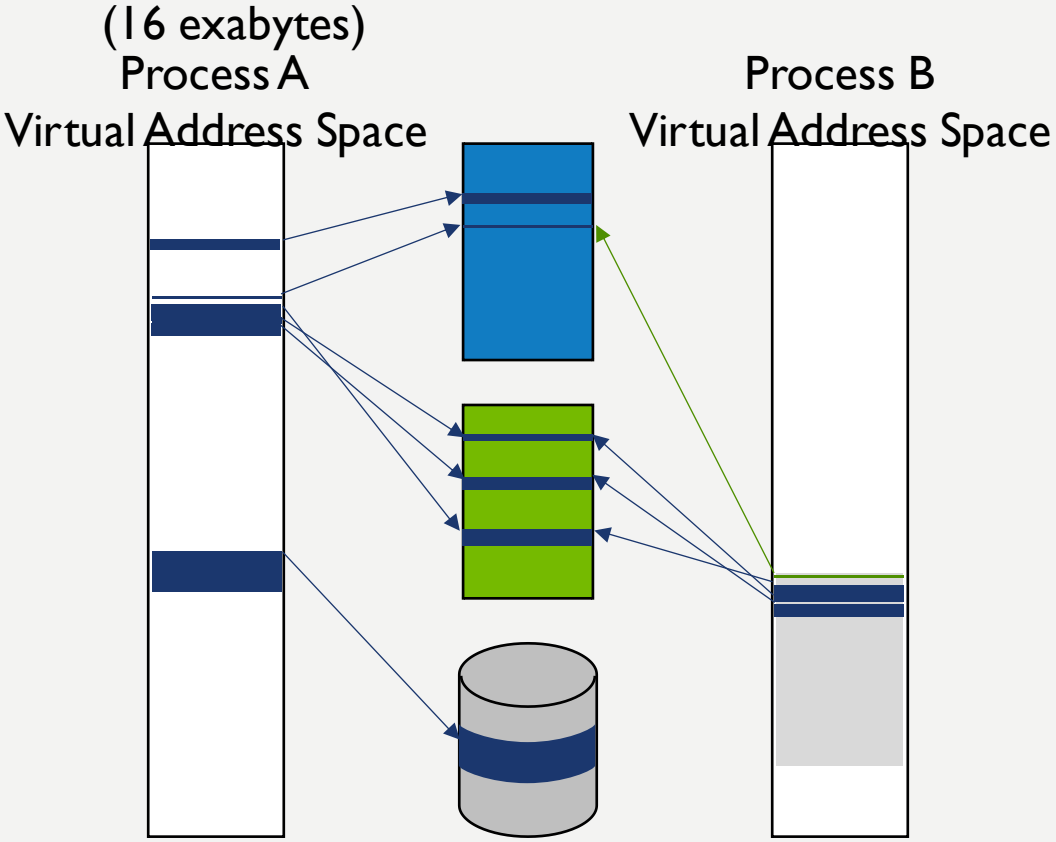
```
class Allocator {
public:
    static const int domain;

    int allocate(size_t size);
    void * convert(void * ptr, int size, const Allocator &alloc);

protected:
    // Can be called from other processes
    void* bootstrap_self();
    void deallocate(int offset);

    void copy_from(void * here, void * there, int size);
    void copy_to(void * here, void * there, int size);

    void copy(void * here, void * there, int size, const Allocator &alloc);

};
```

ALLOCATOR SPECS

- domain
  - Same among allocators in same memory domain
  - Different among allocators not in same memory domain
- allocate
  - Provisions new memory
- deallocate
  - Frees memory
  - Can be called from multiple processes

ALLOCATOR SPECS

- bootstrap
  - Loads allocator into new process
  - Uses virtual memory



ALLOCATOR SPECS

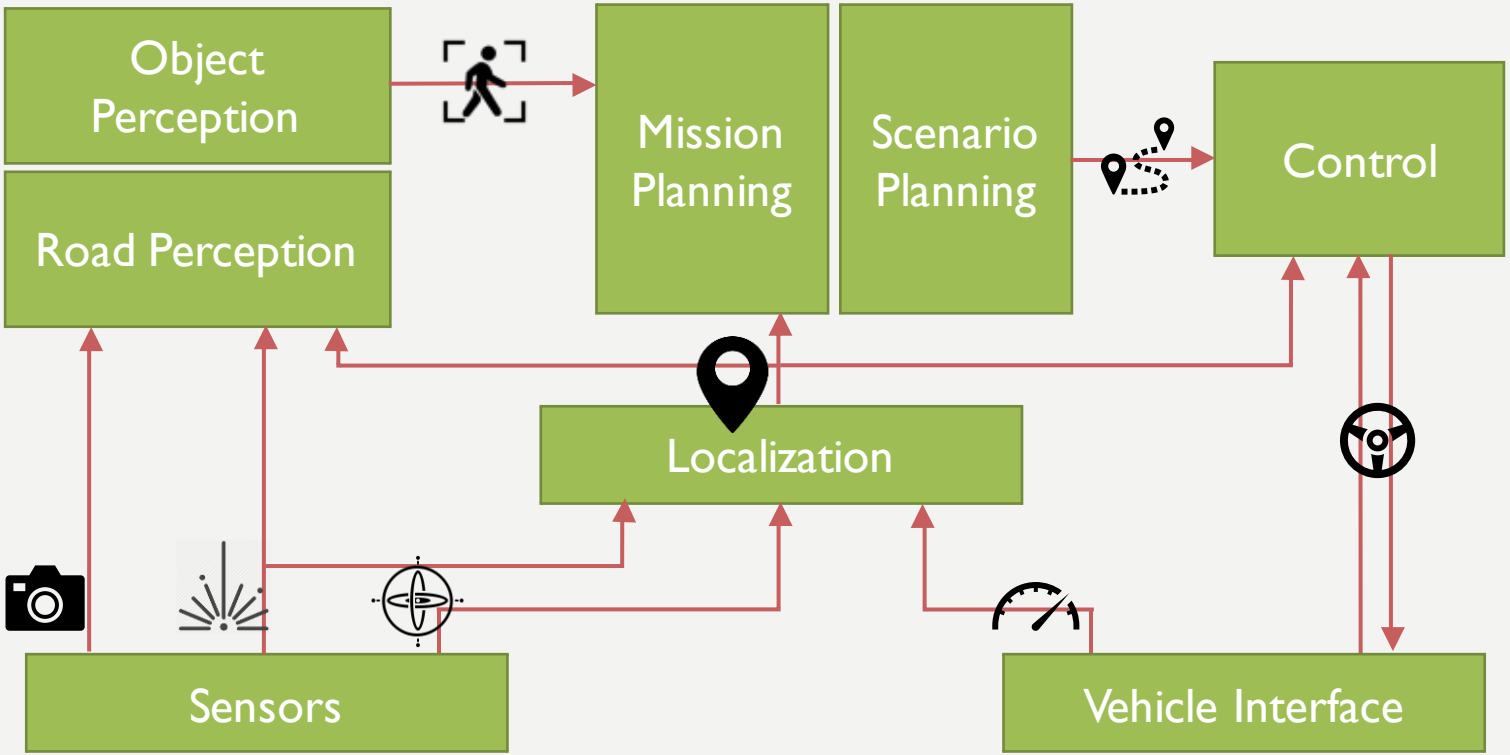
- copy_from
  - How to copy from current domain into main memory
- copy_to
  - How to copy to current domain from main memory
- copy
  - Wrapper for calling two above in sequence
  - Copies device memory to other device memory
    - Uses main memory as intermediate
  - Can be overridden, to bypass main memory

ALLOCATOR SPECS

convert(void * ptr, int size, Allocator &other)

- Converts memory from one allocator to another
- 4 cases, based on memory domain
  - Same to same – pass pointer through w/o copy
  - Device to CPU – call copy_from on other allocator
  - CPU to device – call copy_to on self
  - Device to device – call copy

DESIGN OF A SELF-DRIVING CAR



SELF DRIVING CARS