Subscriber and Data Task FW
Data Distribution

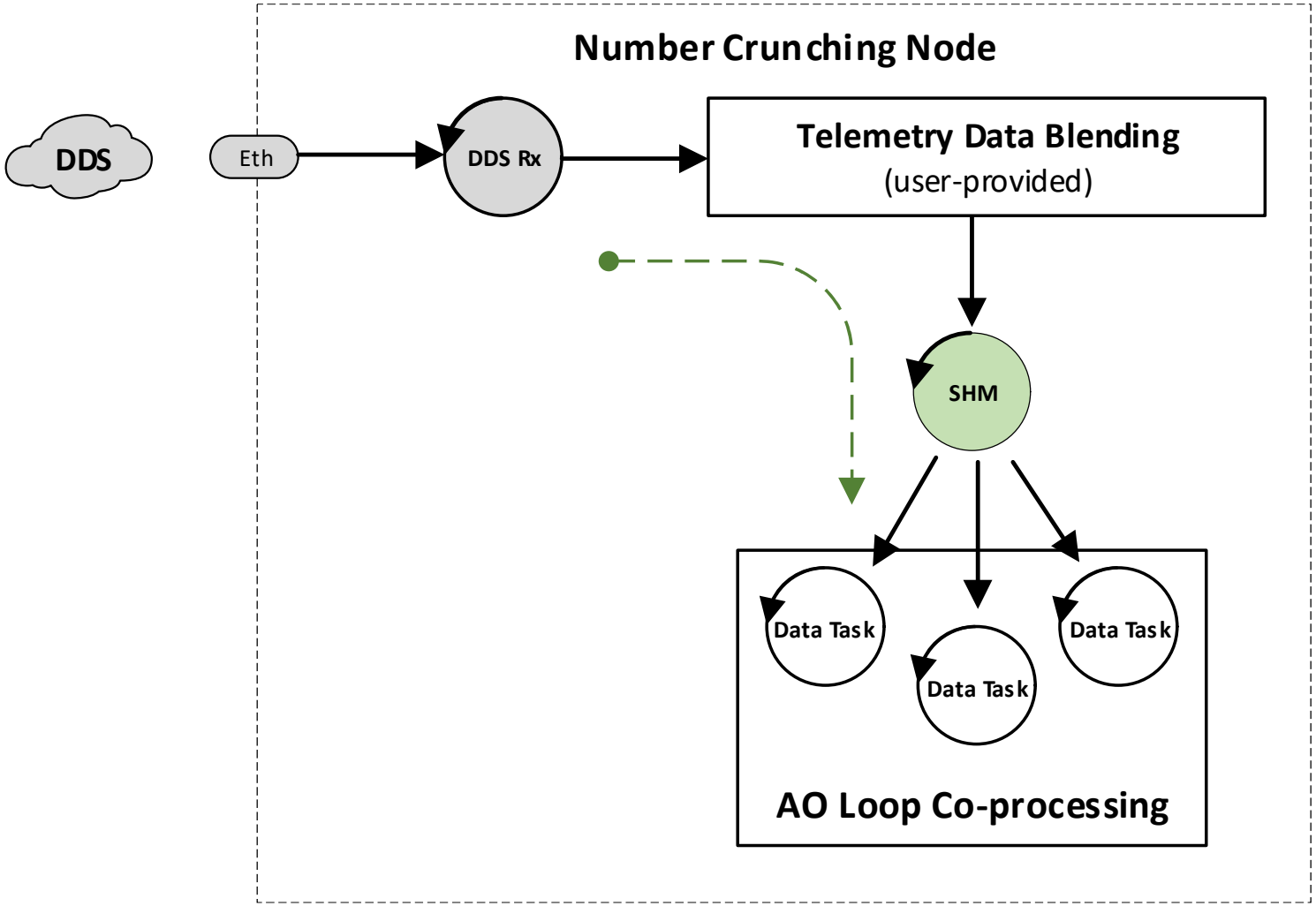
Calle Rosenquist

Subscriber and Data Task FW

Outline

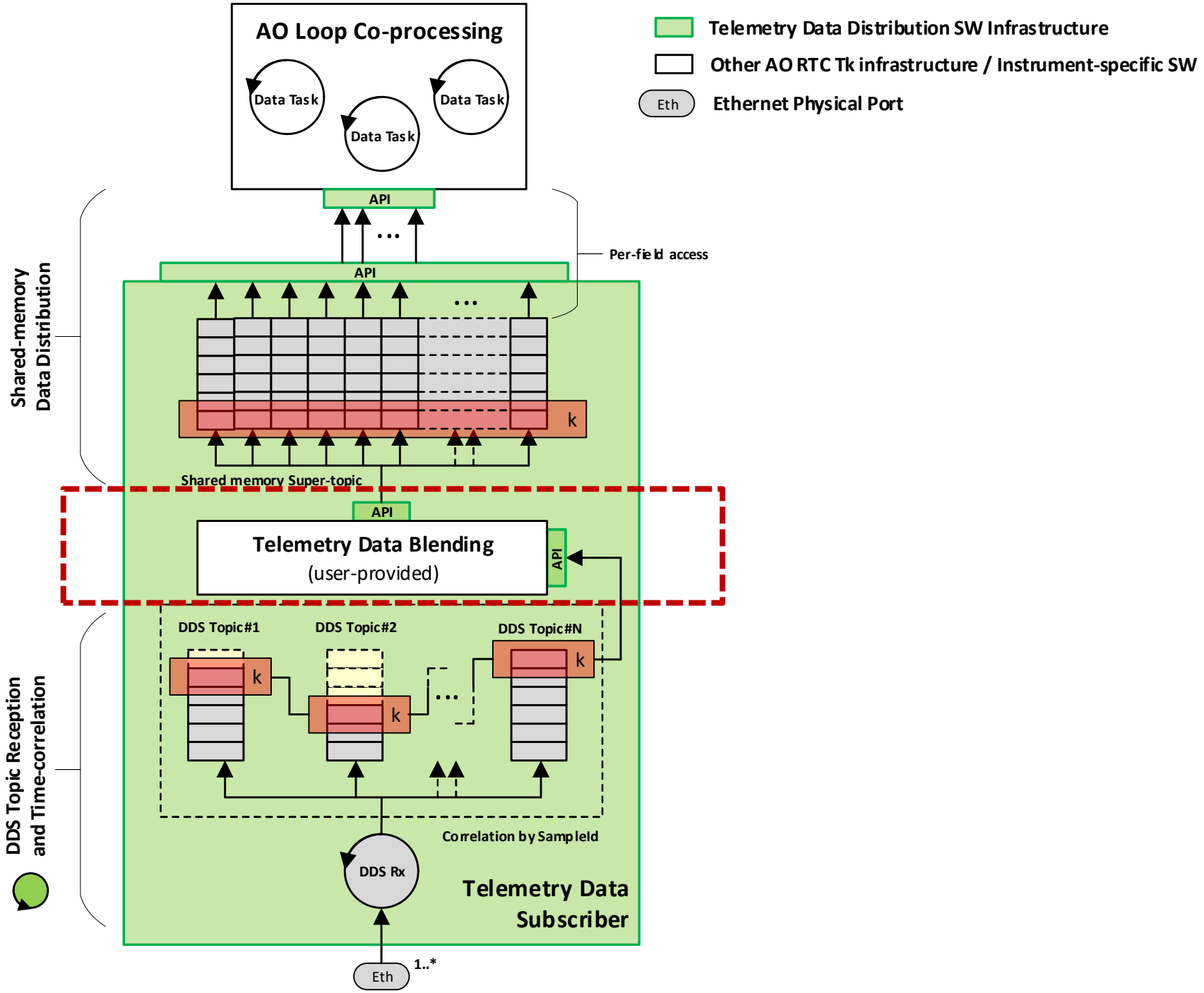
- Data Flow
- Components
- Customization Points
- Prototype Activities

Data Flow


Number Crunching Node
DDS
Eth
DDS Rx
Telemetry Data Blending
(user-provided)
SHM
Data Task
Data Task
Data Task
AO Loop Co-processing

Components

- Telemetry Data Subscriber
  - Configurable Toolkit Component
  - DDS Subscriber
  - SHM Writer with *User Specified Topic Type*
  - User Provided *Telemetry Data Blending*
- Data Tasks
  - User Provided Components
  - Derived from Base Component Infrastructure
  - SHM Reader


AO Loop Co-processing
Data Task
Data Task
Data Task
Telemetry Data Distribution SW Infrastructure
Other AO RTC Tk infrastructure / Instrument-specific SW
Eth
Ethernet Physical Port
API
API
Per-field access
Shared-memory Data Distribution
k
Shared memory Super-topic
API
Telemetry Data Blending
(user-provided)
API
DDS Topic#1
DDS Topic#2
DDS Topic#N
k
k
k
Correlation by SampleId
DDS Rx
Telemetry Data Subscriber
DDS Topic Reception and Time-correlation
Eth
1..*

Telemetry Data Blending

```
// Note: This is an example, API not designed yet!
//
// User provided behaviour in Telemetry Data
// Subscriber: DDS Topic(s) -> SHM Topic
void DataBlending(DdsTopics const& dds_topics,
                  SuperTopic& shm_topic) {
  // populate shm_topic as necessary from dds_topic
  shm_topic.sample_id = dds_topics.sample_id;
  for (auto const& topic : dds_topics.topics) {
    ...
  }

}
```

SHM Topic

Topic Properties

- Trivially Copyable
  - Checked at Build Time

```
// Ok
struct Foo {
  unsigned long sample_id;
  std::array<uint8_t, 128> a;
};

// Not ok
struct Foo {
  unsigned long sample_id;
  std::vector<uint8_t> a;
};
```

SHM Topic

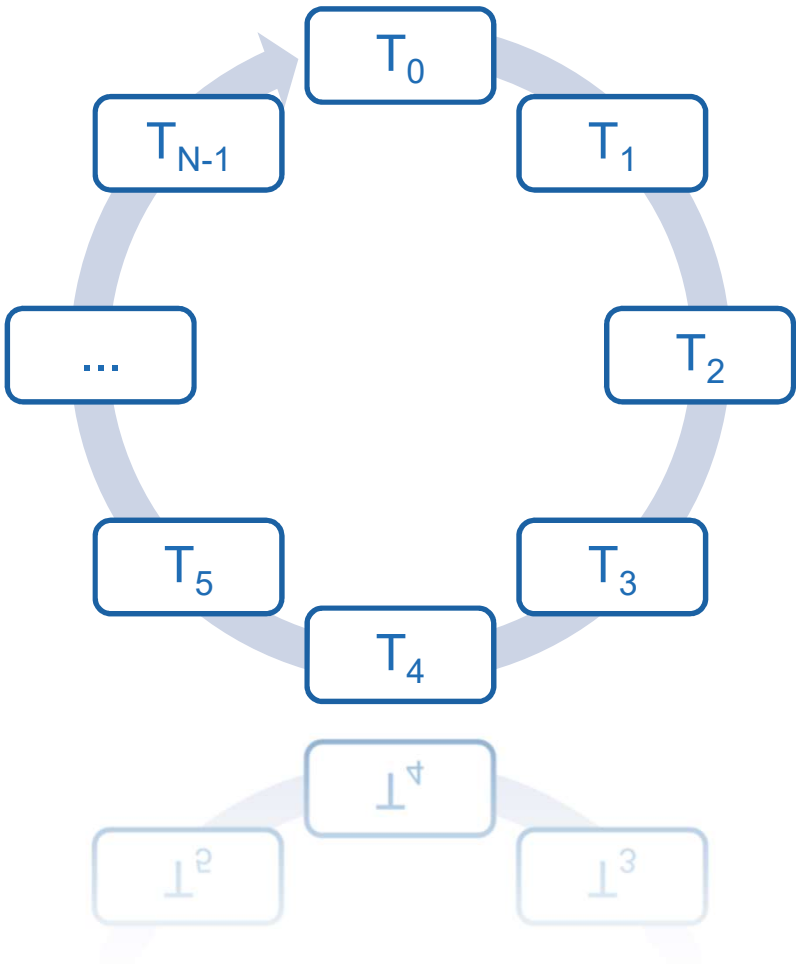
Topic Example

```
struct LgsSensorFrame {
  Timestamp timestamp;
  std::array<float, NUM_GRADIENTS> gradients;
  std::array<float, NUM_SUBAPS> intensities;
};

struct TelemetrySuperTopic {
  unsigned long sample_id;
  std::array<LgsSensorFrame, NUM_SENSORS> sensor_data;
};
```

Shared Memory Queue Prototype

Properties



- FIFO Circular Buffer
  - `mmap(..., MAP_SHARED | ...)`
- Single Writer
- Multiple Readers
- Condition Policies
- Lock free
- Type safe
  - Static type
- Optional Memory Policy
- Configurable Capacity

Shared Memory Queue Prototype
Properties

Provided

- Detection of Invalid Data
- Detection of Lost Data
- No Writer Starvation

Not Provided

- Guaranteed Delivery
- Flow Control
- [Reader] Borrowing/In-place editing

Prototype
Writer

```
// Writer without memory policy
auto writer = Writer<TelemetrySuperTopic>("topicname", 300);

// Create memory policy to bind memory to NUMA node 0:
auto mempol = MemPolicy::MakeBindNode(0);
auto writer = Writer<TelemetrySuperTopic>("topicname",
                                          300,
                                          mempol);


// Single write
TelemetrySuperTopic t;
...
auto err = writer.Write(t, true);

// Batch write
std::vector<TelemetrySuperTopic> ts;
auto err = writer.Write(ts.data(), ts.size(), true);
```

Application Data Extraction: Full topic

```
// Attach to existing queue
auto reader = Reader<TelemetrySuperTopic>("topicname");

// Single read
TelemetrySuperTopic t;
auto err = reader.Read(t, 2s);
if (err) {
  // Read failed
  ...
}
// Batch read
std::array<TelemetrySuperTopic, 10> ts;
auto [err, count] = reader.ReadSome(ts.data(), ts.size(), 2s);
```

Application Data Extraction: Subset

```
auto reader = Reader<TelemetrySuperTopic>...;

// Extract parts
std::array<float, NUM_GRADIENTS> gradients;
auto err = reader.Read(gradients, 2s);
```

```
void Transform(TelemetrySuperTopic const& topic,
               std::array<float, NUM_GRADIENTS>& subset) {
  subset = topic.sensor_data[0].gradients;
}};
```

Prototype
Reader

Application Data Extraction

```
auto reader = Reader<TelemetrySuperTopic>...;

// Extract parts
struct Refs {
  uint8_t& data;
  ...
};
auto err = reader.Read(Refs(mem_ref), 2s);

void Transform(TelemetrySuperTopic const& topic,
               Refs& refs) {
  std::memcpy(refs.data,
              topic[0].gradients.data(),
              topic[0].gradients.size());
}};
```

Prototype
Results

- Most Demanding Test
  - AMD EPYC 7551, 32-core, 8 NUMA nodes
  - Topic size 55 kiB, queue capacity 700 elements
  - Writer
    - NUMA 0
    - Fills topic with sequence number
  - Readers
    - 18 in total, 3 per NUMA node 2-7 (localalloc)
    - Reads through the whole topic and verifies sequence numbers
- Result
  - ~ 17 kHz avg @ batch size 10 with atomic spin condition
  - ~ 14 kHz avg @ batch size 10 with pthread condition variable
    - < 3 kHz @ batch size 1

Prototype
Future Work

- Add skipping
- Improve spin condition with back-off
- Provide Borrowing Interface for Writer
  - Enables work-in-place
- Investigate alternative reader API using callbacks

```
Reader::Read([&](TelemetrySuperTopic const& topic) {
  std::memcpy(topic...);
}, 2s);
```

Questions?