The Kernel

The Kernel

Scheduler: Round-Robin, Priority Queues, Tree Flavours
Scheduler Actors: Features, Timers, Async I/O
Streams Backends: Zero-copy, Message Passing
Linear Backends: Async I/O Disk Streams, Network Streams
Indexed Backends: Timers, Actors
Backpressured Message Bus/Buffers: Arc/Vec prealloc
Class: Low Latency, Real Time

Respect Kernels History

Richard Rashid. Mach 3. NUMA, Bus Oriented Components
Dave Cutler. Windows NT. True Async I/O on IoCompletionPort
BeOS. Travis Geiselbrecht. SMP Scaling of OS services
Microkernels: RT, Tiny codebases eCos/TRON, QNX, VxWorks
Unikernels: Erlang, Mirage, HaLVM

FOUNDATION

Stream/List duality

```
pub enum List<Message> {
            Nil,
            Cons(Message, std::marker::PhantomData<List<Message>>) }

pub struct Stream<Message> {
      head: Message,
        tail: Box<Stream<Message>> }

  pub trait Stream<Message> {
          fn head(&mut self) -> Message;
          fn tail(&mut self) -> Stream<Message>; }
```

FUTURES

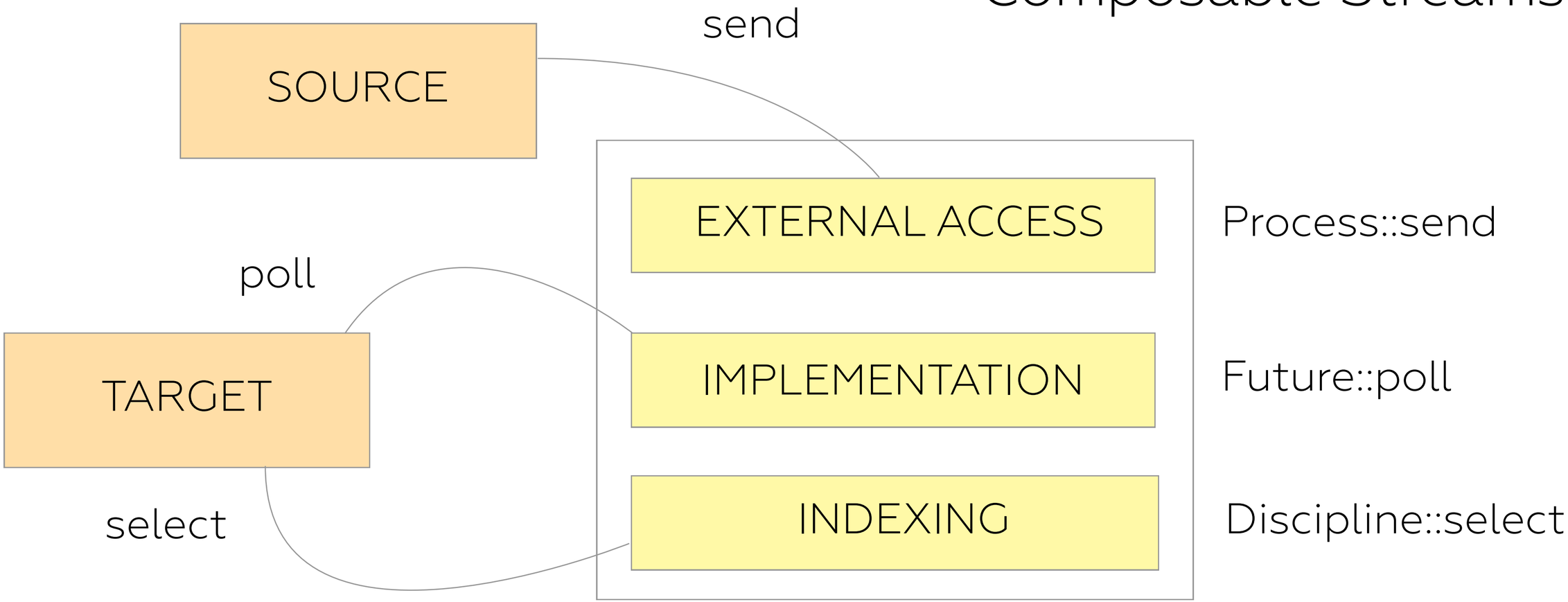
Zero-copy and Message Passing

```
pub trait Future<Message,Error> {
        fn poll(&mut self) -> Result<Message,Error>;
        fn tail(&mut self) -> Future<Message,Error>; }

pub trait Process<Protocol, State, Error> {
        fn state(&mut self) -> State;
        fn send(&mut self, Protocol) -> Result<State, Error>; }

pub trait Discipline<Stream<Message>> {
        fn select(&mut self, u64) -> Stream<Message>; }
```

Composable Streams



REACTOR

Queue Disciplines

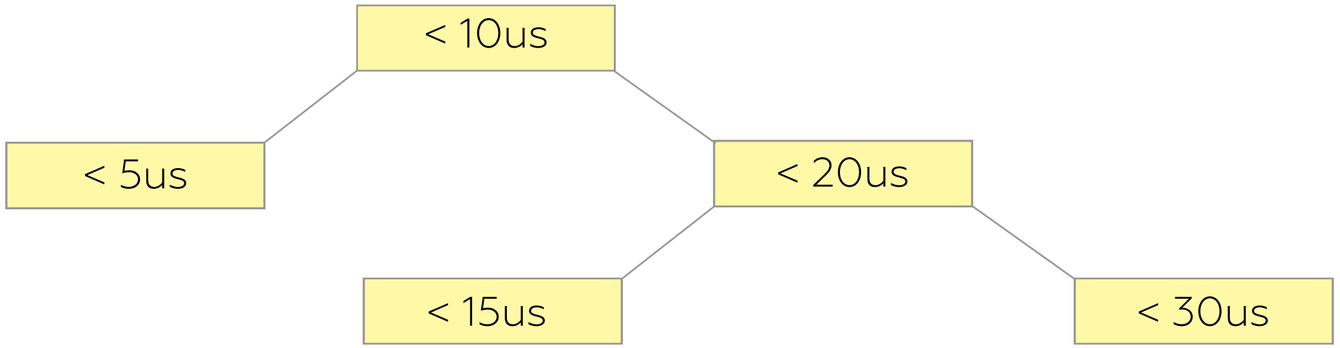
I/O

Monotonic Sequence

Circular Buffers

TIMERS

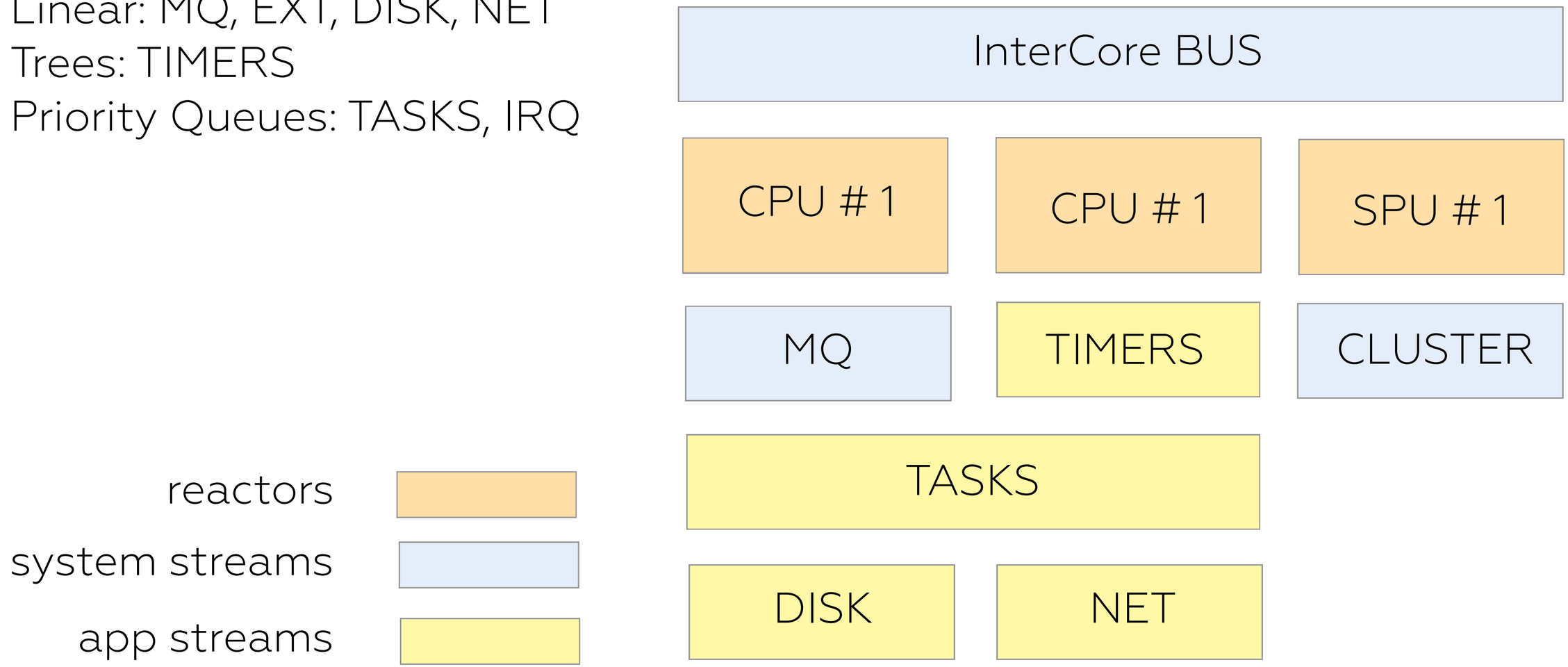
Low-latency Tree



TASKS

Priority Queues

Linear: MQ, EXT, DISK, NET
Trees: TIMERS
Priority Queues: TASKS, IRQ



REACTOR

Discipline::select

TASKS

Core Context

TOKIO

REVENT

Single Task Tree per Core

NANOMSG

Scheduler::select

RX

Reactor Requested Features

1. Reactor Core Context
2. Task Context
3. Scheduleable Entities
4. Task List
5. Task Selection by Priority
6. API: spawn/select/poll
7. Inspirational Libraries: Rx, Nanomsg, Tokio

MQ

Vec, VecDeque, Link, Turbine, Pipes

Queue Types

(rx , tx) CHANNEL

Discipline::select

SPSC/LINK

MPSC/SUB

SPMC/PUB

Non-schedulable Array Store

Queues Requested Features

1. Three Basic Drivers: SPSC, MPSC, SPSC
2. Different Backends: Vec, VecDeque, Link, Turbine, Pipes
3. Queue Properties
4. Queue with Priorities
5. API: create/pub/sub/link
6. Inspirational Libraries: Turbine, Rust Queues, Pipes

TIMERS

Discipline::select

TIMER

(interval , task_id)

TOKIO

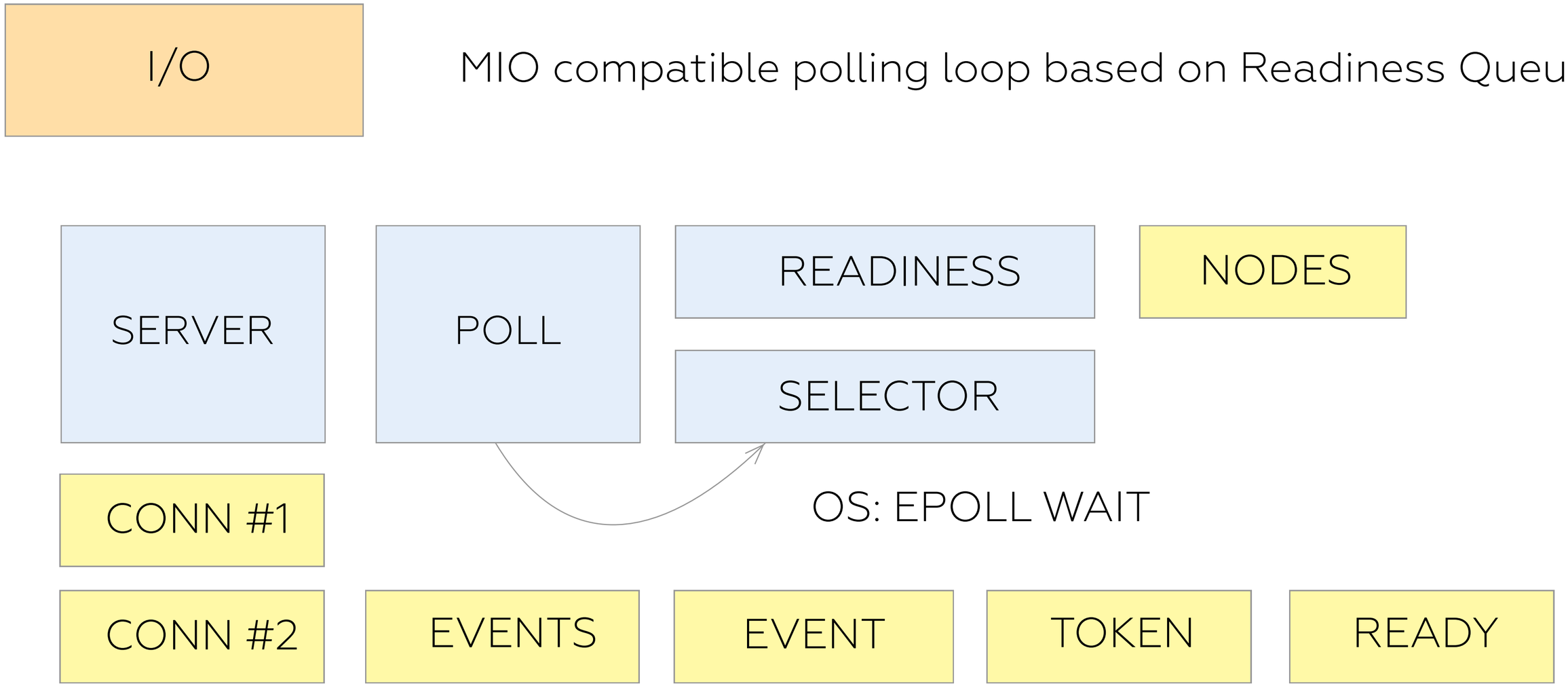
Dedicated Core or Interrupt Queue

Scheduler::select

RX/RUST

Timers Requested Features

1. Timers Core Context
2. Timer Types: Oneshot Scalable Timers, Fixed List of Interval Timers
3. API: create/select/poll
4. Inspirational Libraries: Tokio, Rx


I/O
MIO compatible polling loop based on Readiness Queue
SERVER
POLL
READINESS
NODES
SELECTOR
CONN #1
OS: EPOLL WAIT
CONN #2
EVENTS
EVENT
TOKEN
READY

Network I/O Features

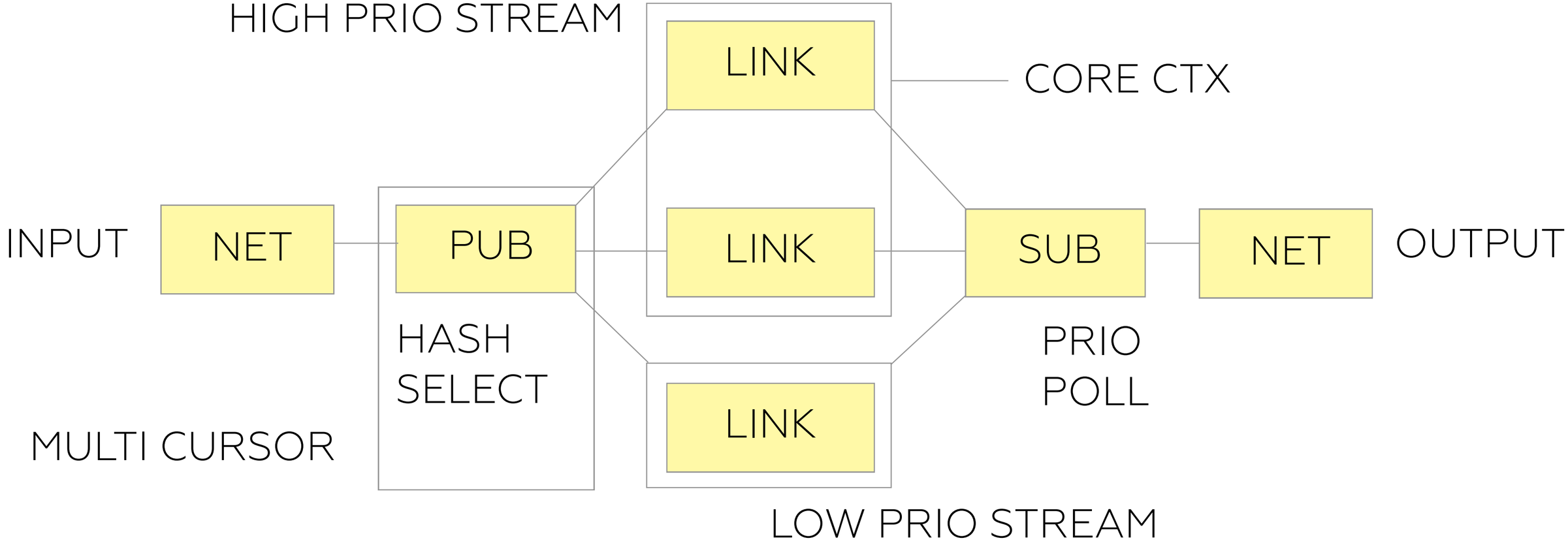
1. MIO compatible API
2. Evented Polling on Linux, BSD, XSOCK
3. Connection State
4. API: create/select/poll
5. Close to FFI API
6. Client and Server Samples
7. Tele and Command Stdin Streams

```
$ . ../cmplxt.sh

struct    32
trait     2
impl      85
```

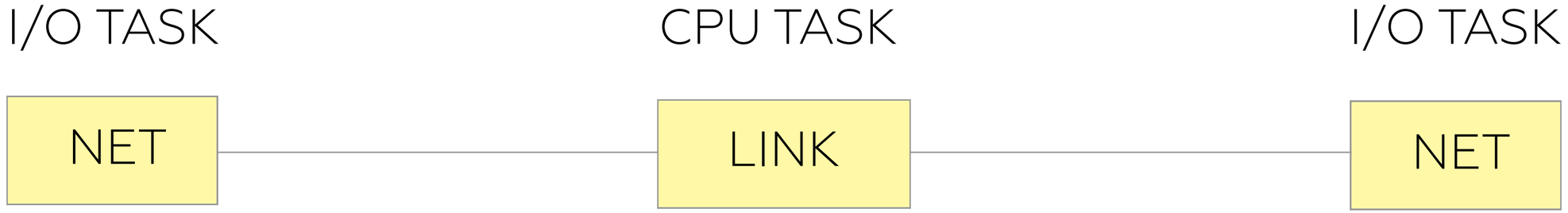
LOAD BALANCING CASE

Load Balancing
of Priority Streams per Core Buckets



FAST DELIVERY CASE

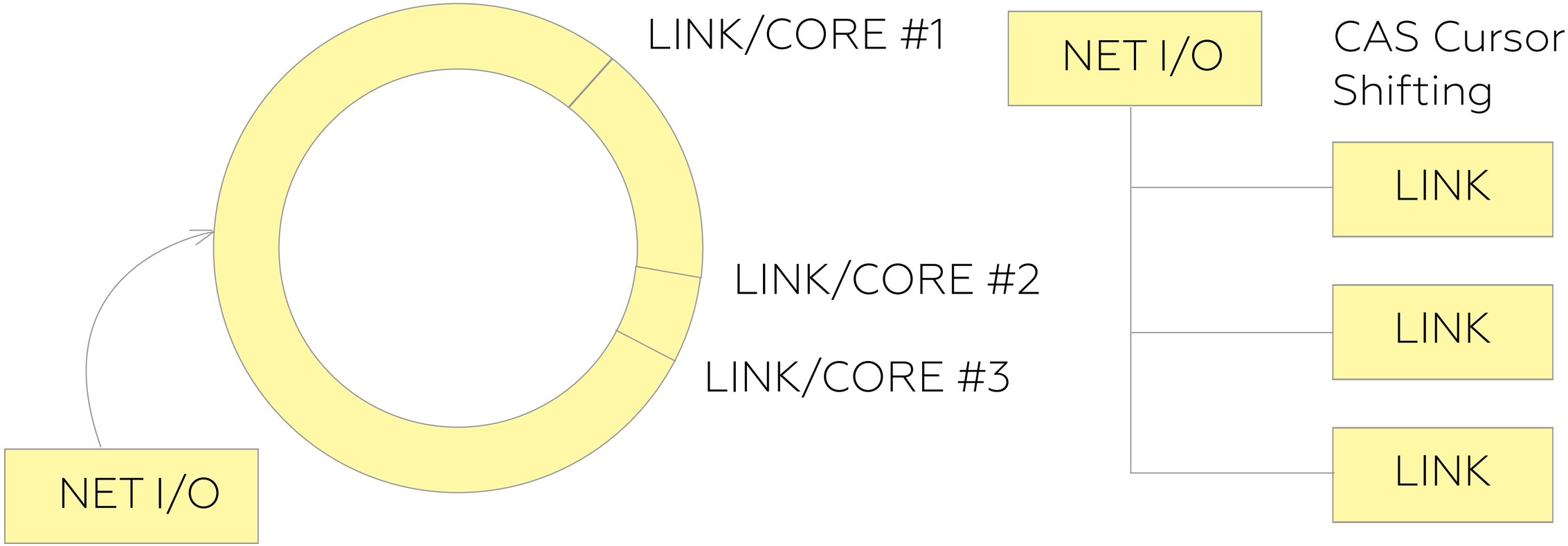
Single Threaded Task Configuration
to be compared as reference



You can use inplace message modifying and reduce copies to unpack and pack.

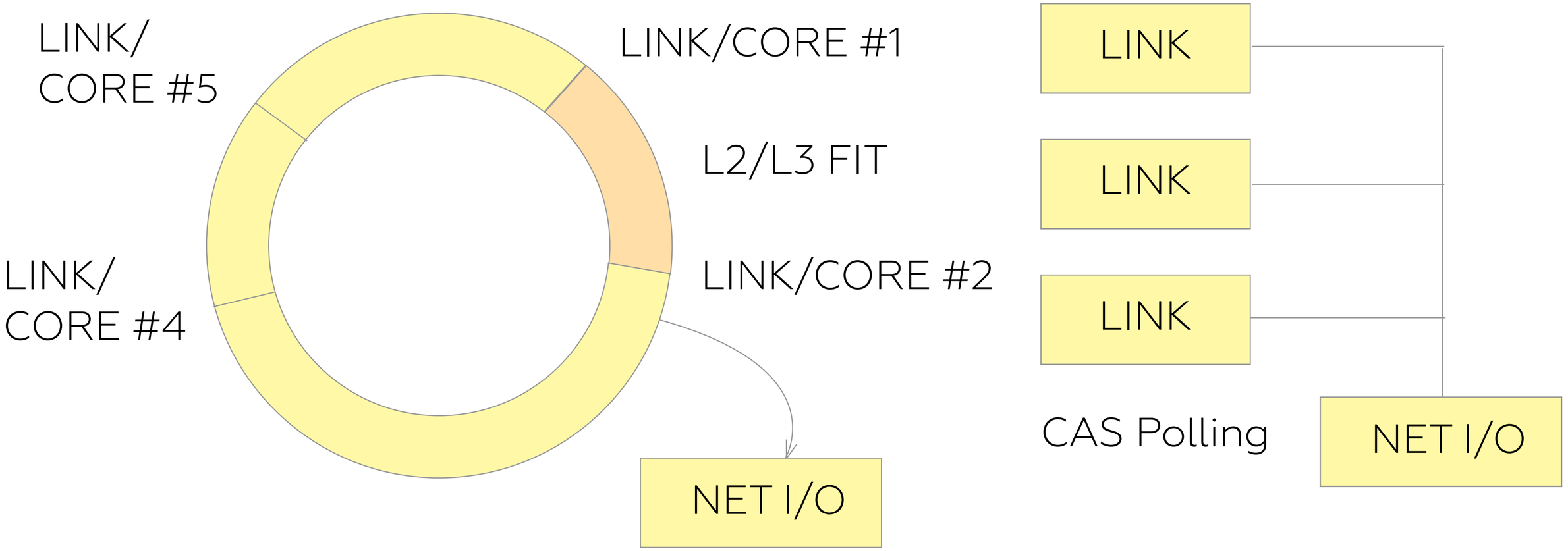
PUBLISHER CASE

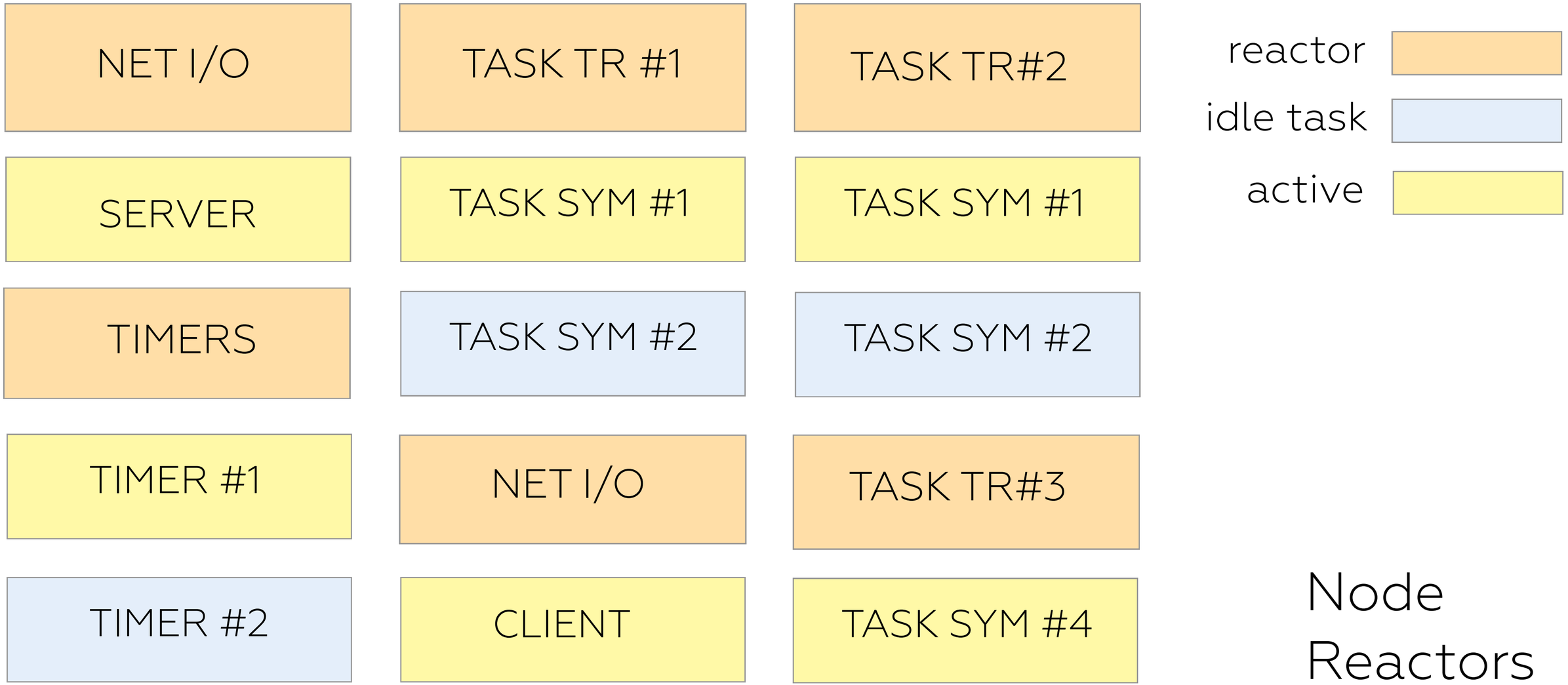
PUB Implementation for Zero-Copy Multiple Consumer Publishing (SPMC)



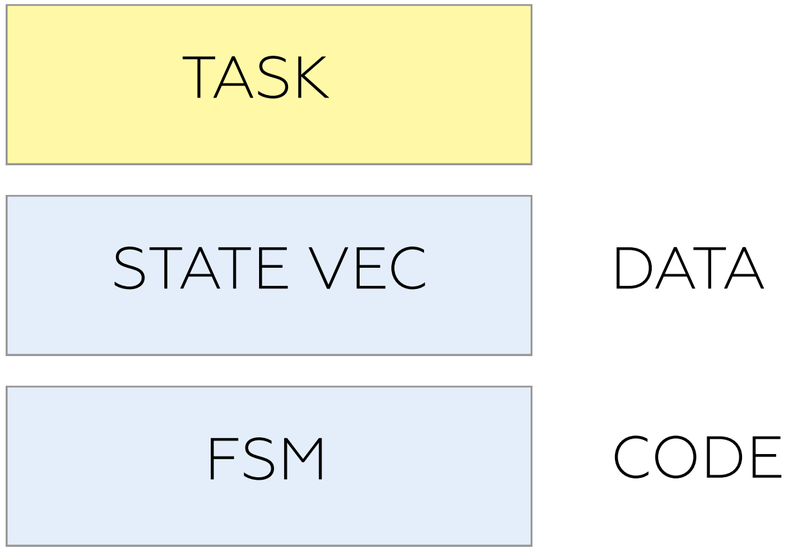
SUBSCRIBER CASE

Multicursor Implementation of SUB (MPSC) for InterCore Queue Migrations and Cache Locality




NET I/O
TASK TR #1
TASK TR#2
SERVER
TASK SYM #1
TASK SYM #1
TIMERS
TASK SYM #2
TASK SYM #2
TIMER #1
NET I/O
TASK TR#3
TIMER #2
CLIENT
TASK SYM #4
reactor
idle task
active
Node
Reactors

Task Taxonomy

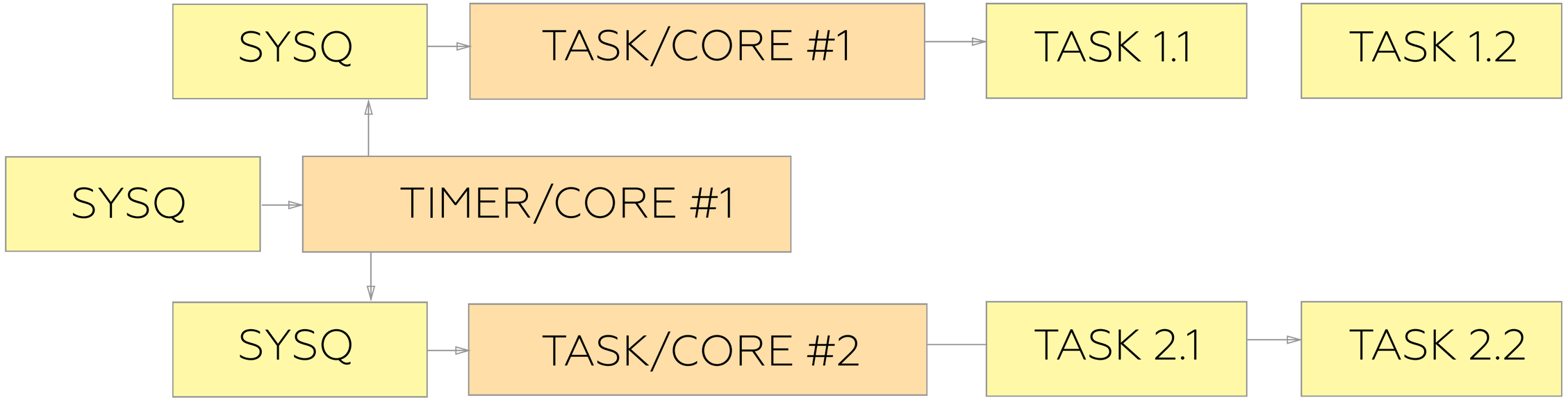


Object Table: Cursors, Counters

CUR #1 R/W	0—0xFFFF
CUR #2 R	0xFFFF—0xFFFF0000
CUR #3 W	0xFFFF0000—0xFFFFFFFF
CNT #1	00120090912090

INTERCORE QUEUES

Scheduler Reactors can communicate throught InterCore transport when needed.



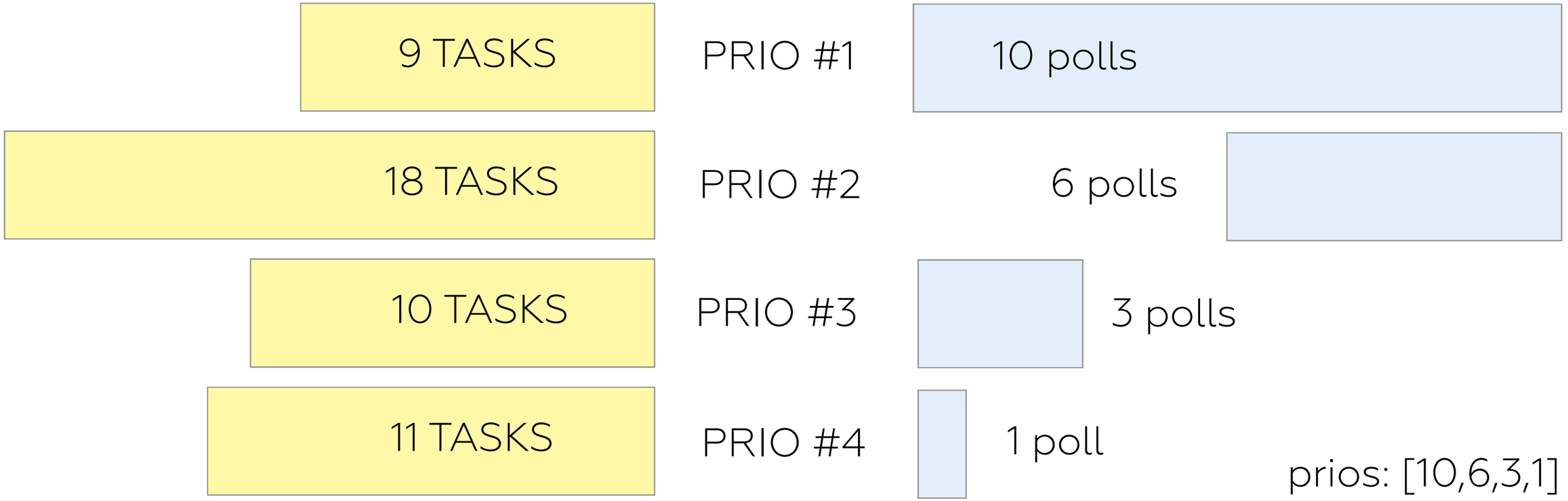
Network Polling also could be seen
as InterCore single sourced Queue for Task Delivery.

InterCore Protocol

io	seq	ring
register	spawn	join
send	cursor	split
sync	reactor	timer

Workload: 48
Time: 20

Capacity: 239
Total: 400



prios: [10,6,3,1]

Times * Types * Polls = Capacity

Time in [0,MaxTime]
Type, Poll in const