



Ensō Operating System

# 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>; }  

```

## ORDERS

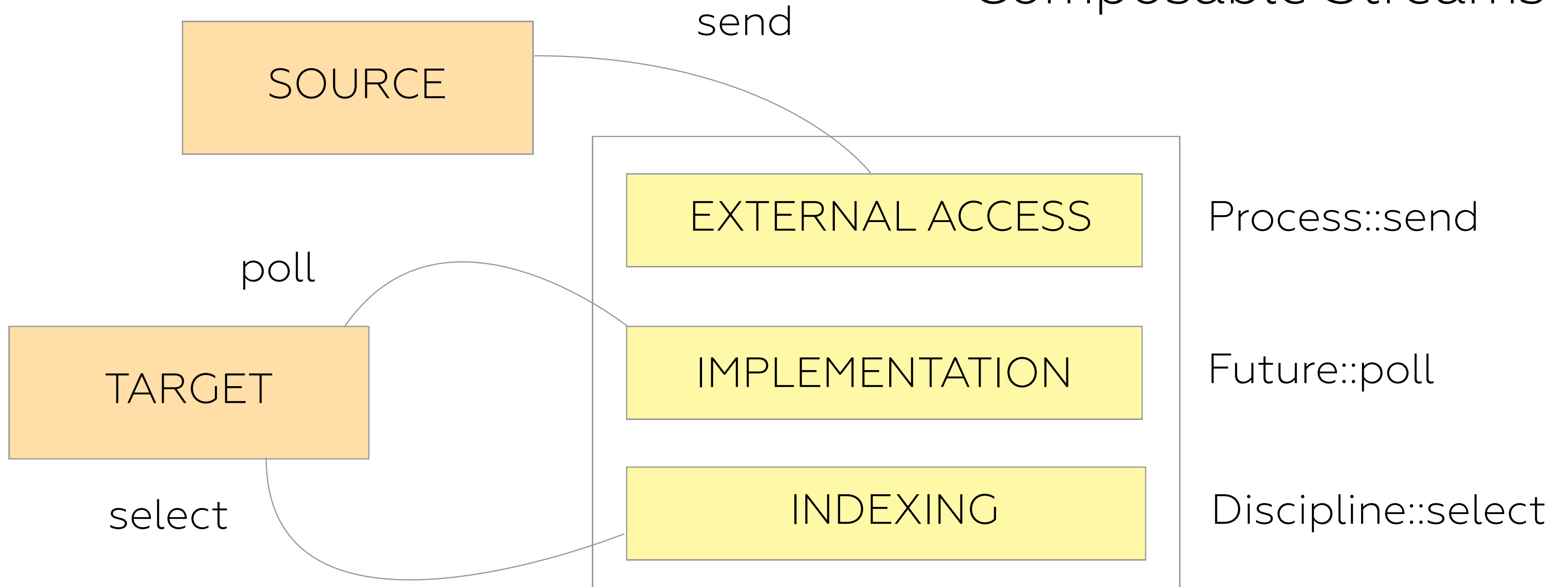
## Order Processing Protocol

```
pub struct OrderState {  
    state: OrderStatus,  
    id: ID,  
    price: Price,  
    size: Size,  
    side: Side, }
```

```
pub enum OrderProtocol {  
    Request, Execute, Reject,  
    UnsolicitedCancel,  
    Cancel,  
    Ack, Replace, }
```

```
type OrderProcess = Process<OrderProtocol, OrderState, Error>;
```

# Composable Streams



# Polymorphic Disciplines

SCHED

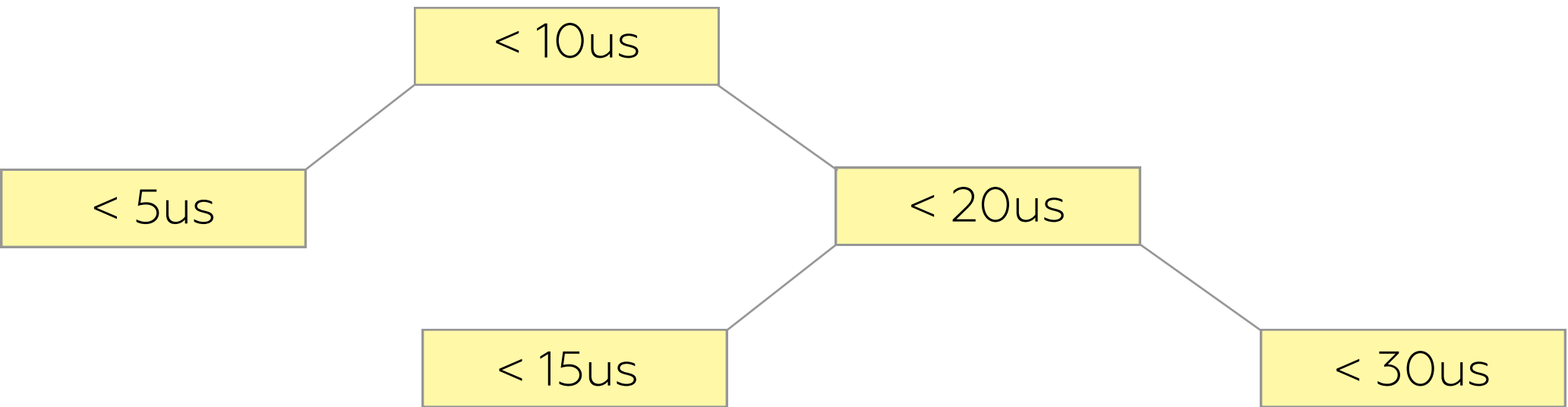
DISK

Discipline::select

Circular Buffers

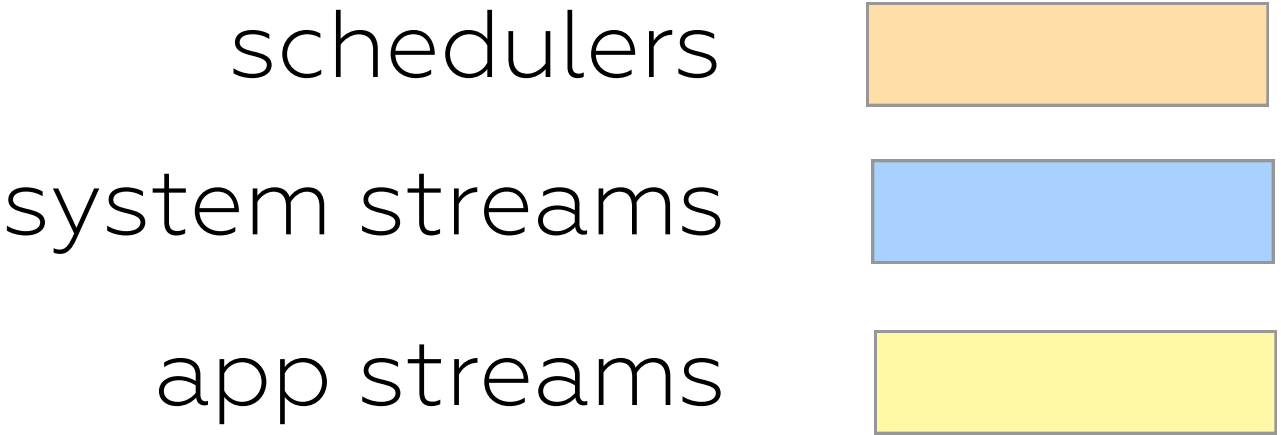
TIMERS

Discipline::select

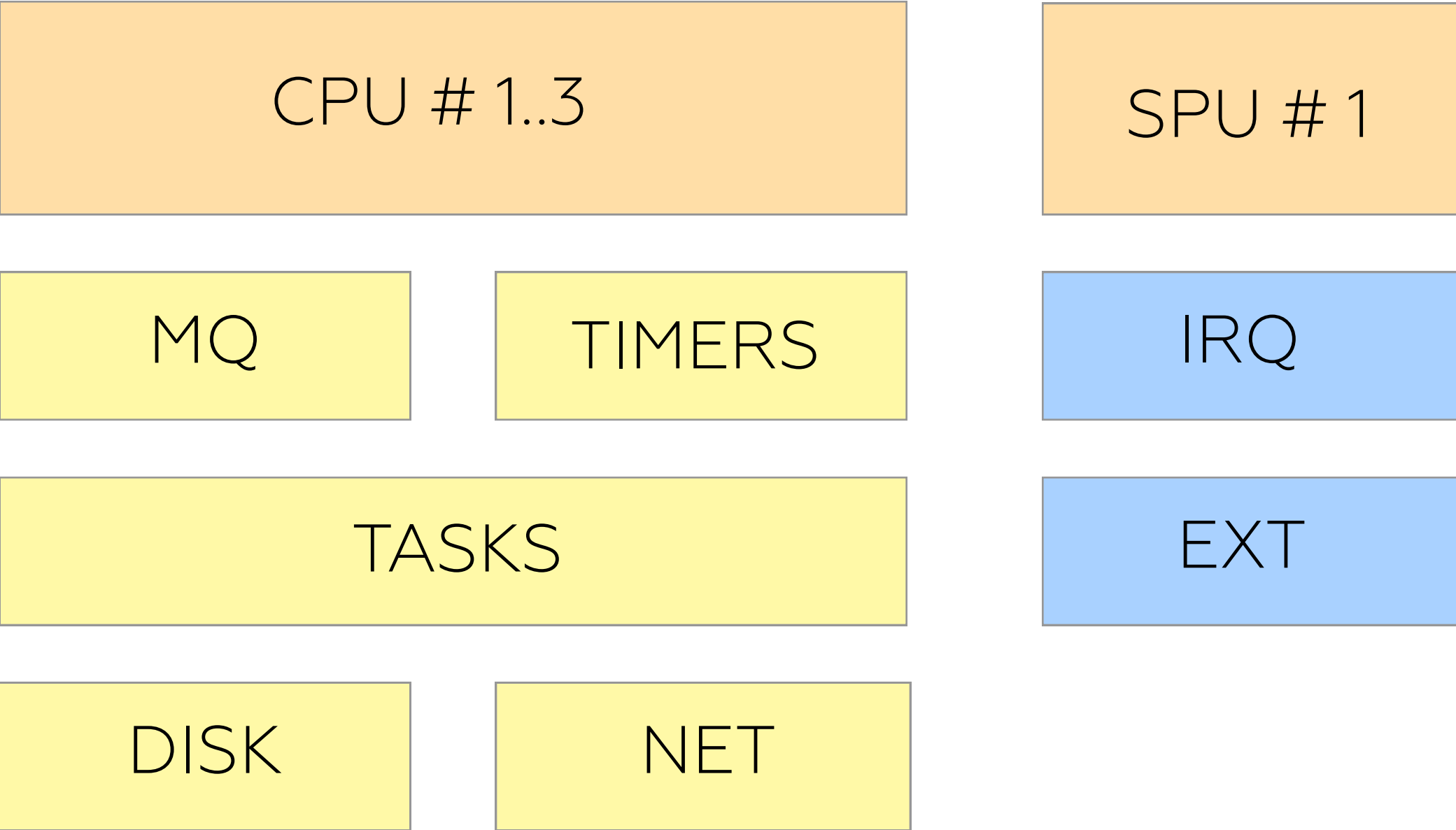




Linear: MQ, EXT, DISK, NET  
Trees: TIMERS  
Priority Queues: TASKS, IRQ



# Node Components



REACTOR

Discipline::select

Core Context

TASKS

TOKIO

REVENT

Single Task Tree per Core

NANOMSG

Scheduler::select

RX/RUST

# 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

## Queue Types

SPSC/LINK

MPSC/SUB

SPMC/PUB

Vec, VecDeque, Link, Turbine, Pipes

( rx , tx )

CHANNEL

Discipline::select

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

NET I/O

MIO compatible polling loop based on Readiness Queue

SERVER

POLL

READINESS

NODES

SELECTOR

CONN #1

CONN #2

EVENTS

EVENT

TOKEN

READY



OS: EPOLL WAIT



# Network I/O Features

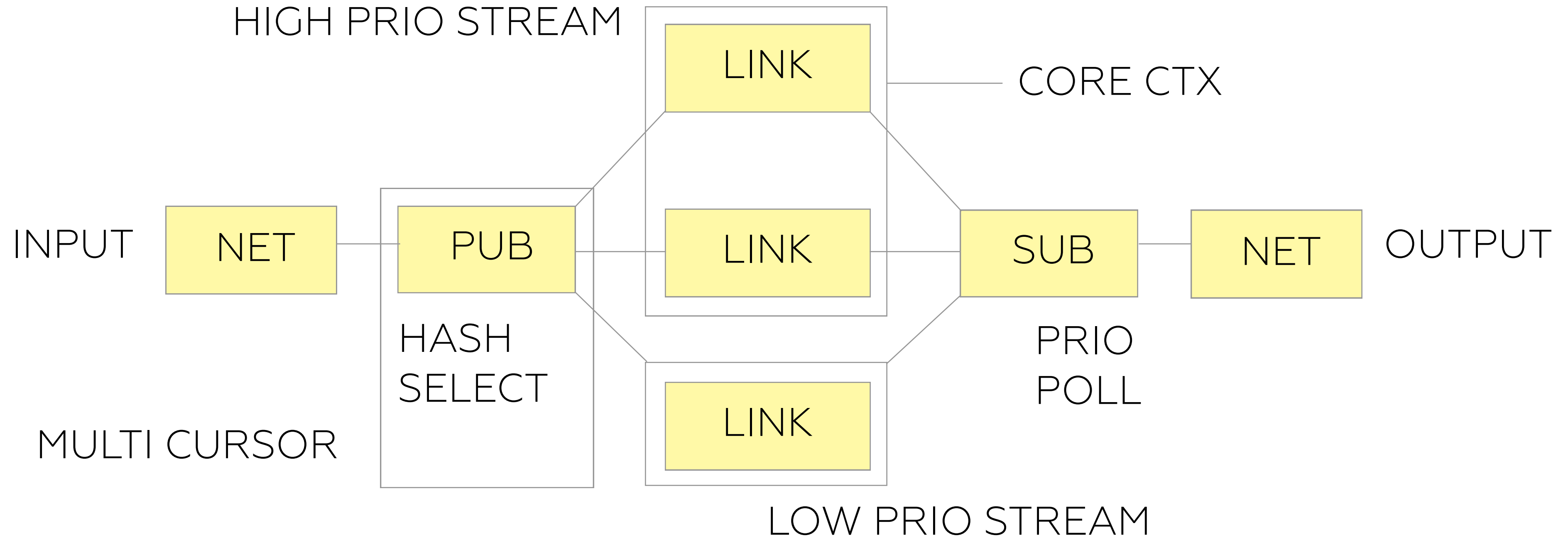
1. MIO compatible API
2. Evented Polling on Linux, BSD, XSOCK
3. Connection State
4. API: create/select/poll
5. Close to FFI API
6. Client and Server Samples

```
$ . ../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

I/O TASK



CPU TASK



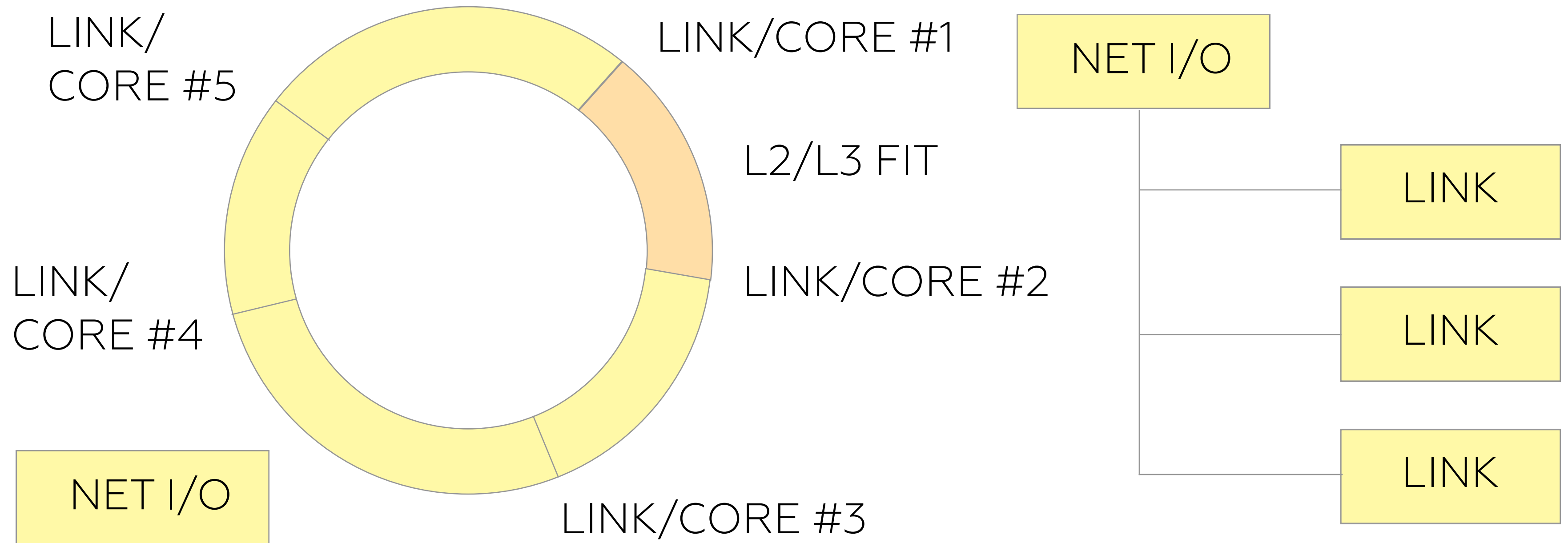
I/O TASK



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

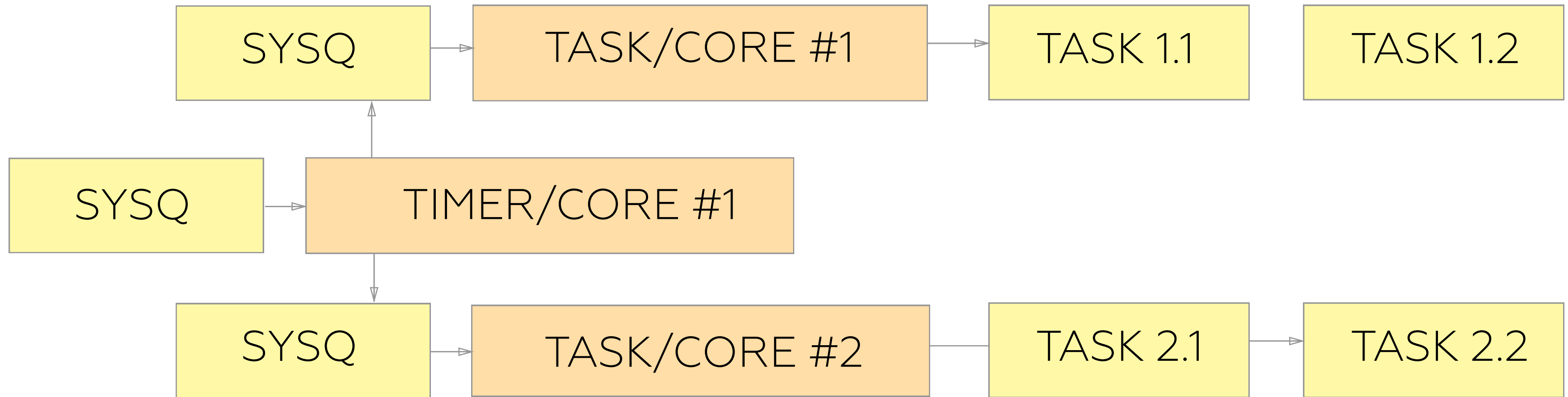
# PUBLISHER CASE

# Multicursor Implementation of PUB for InterCore Queue Migrations and Cache Locality



## INTERCORE QUEUES

Scheduler Reactors can communicate through InterCore transport when needed.



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