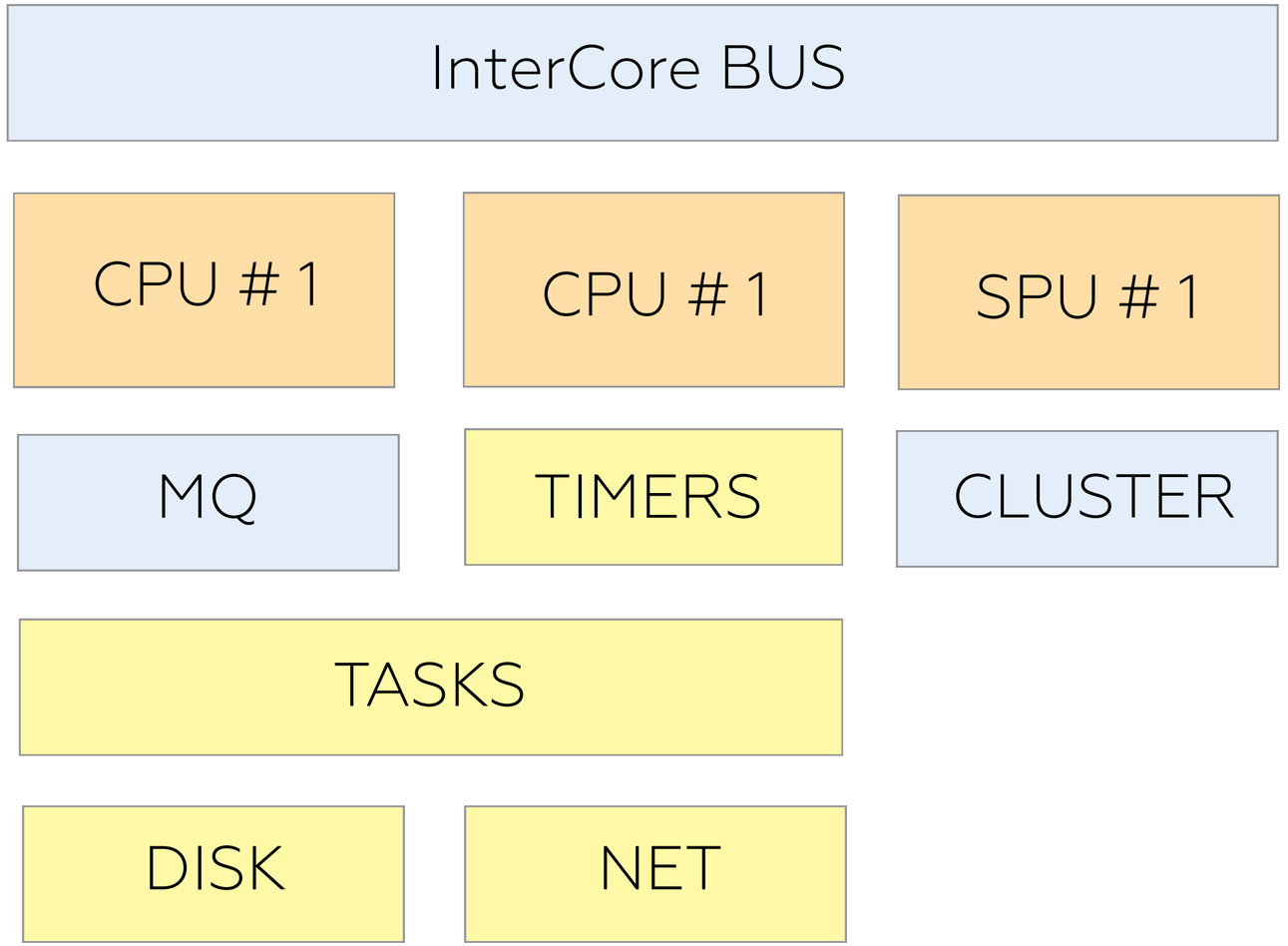
The Kernel

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

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



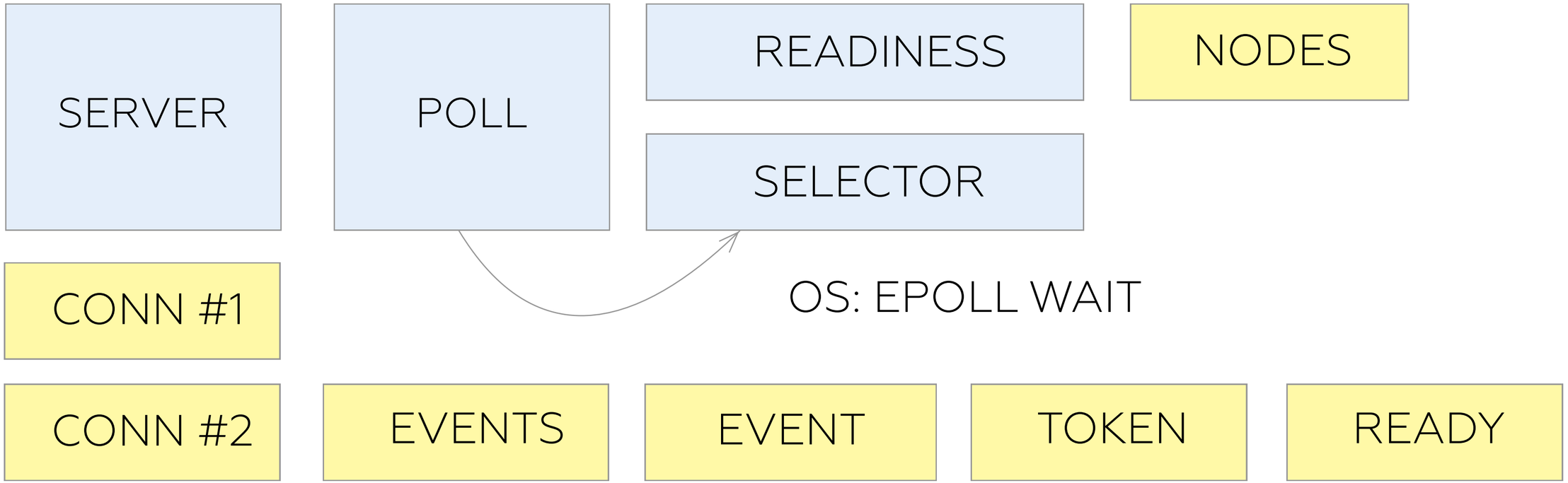
reactors

system streams

app streams

IO

MIO compatible polling loop based on Readiness Queue



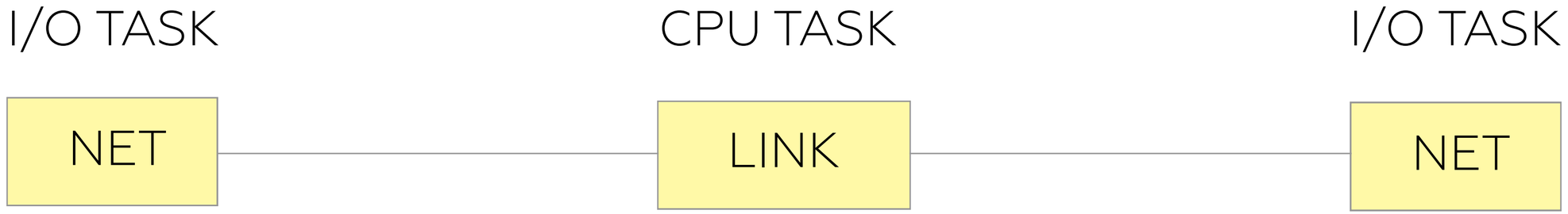
MQ

Queue Types

SPSC/LINK	4-10ns Lowest Latency Possible
MPSC/SUB	10-40ns Reducer or Subscribe Polling
SPMC/PUB	10-40ns Publisher Multicursor

FAST DELIVERY CASE

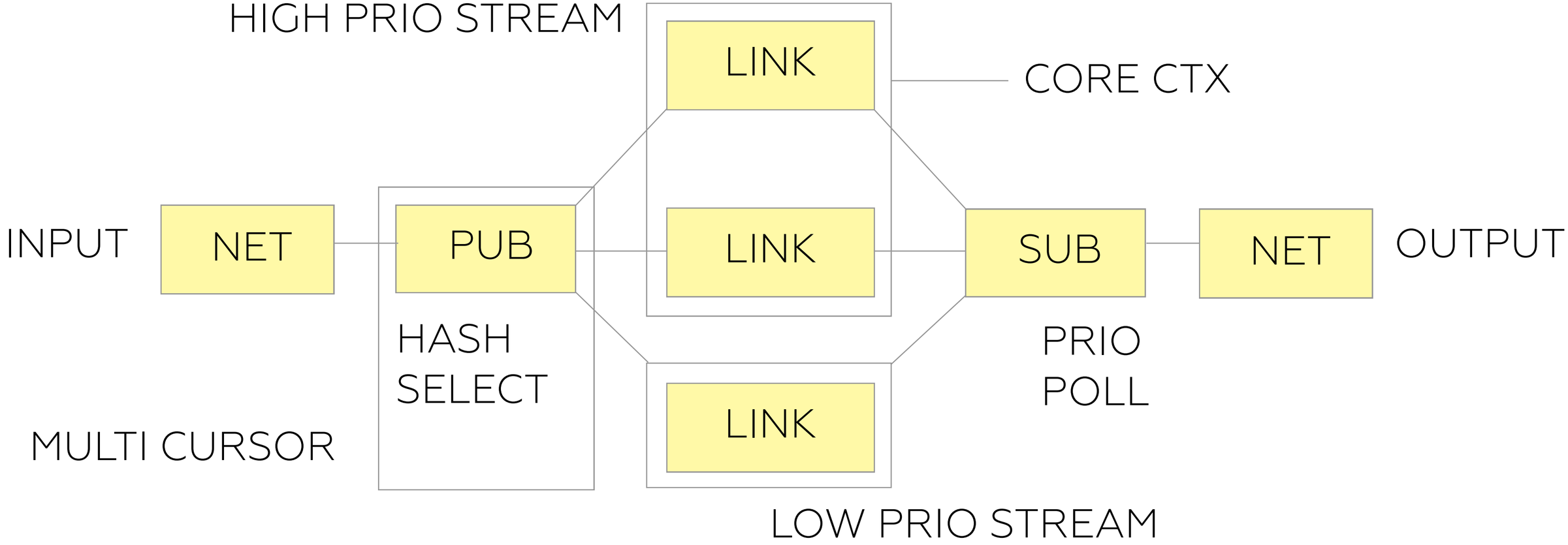
Single Threaded Task Configuration
to be compared as reference



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

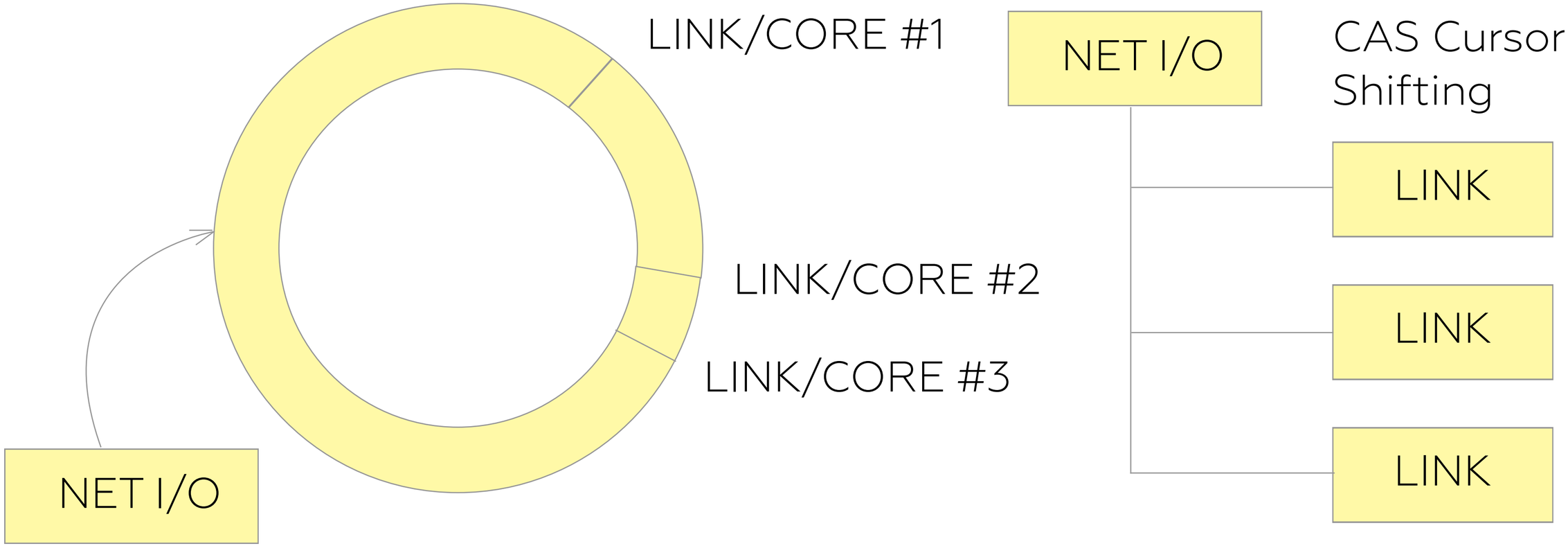
LOAD BALANCING CASE

Load Balancing
of Priority Streams per Core
Buckets



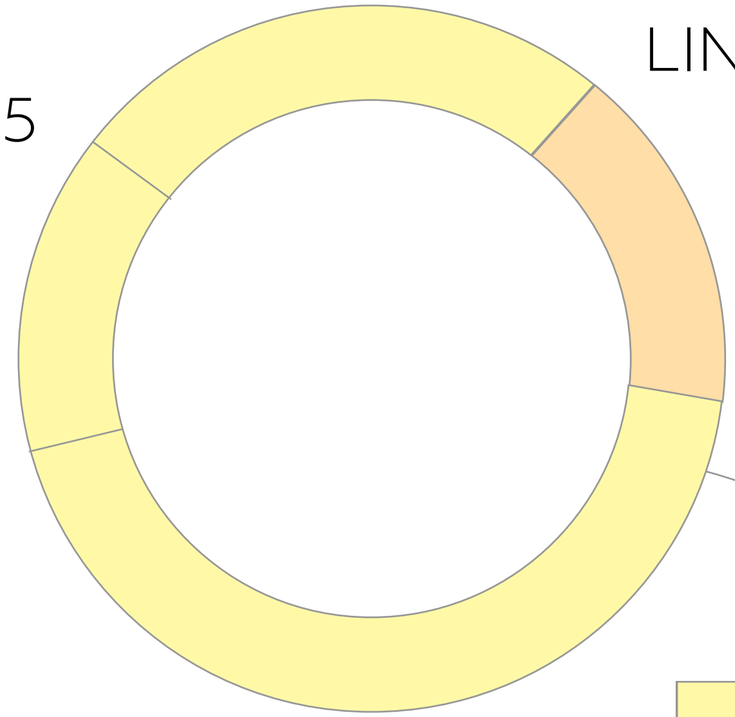
PUBLISHER CASE

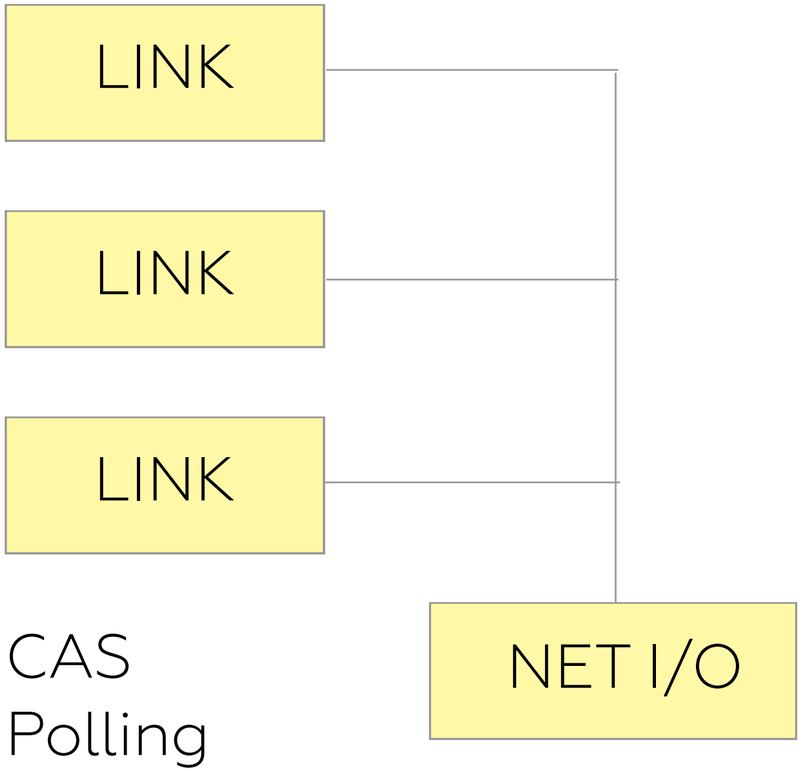
PUB Implementation for Zero-Copy Multiple Consumer Publishing (SPMC)



SUBSCRIBER CASE

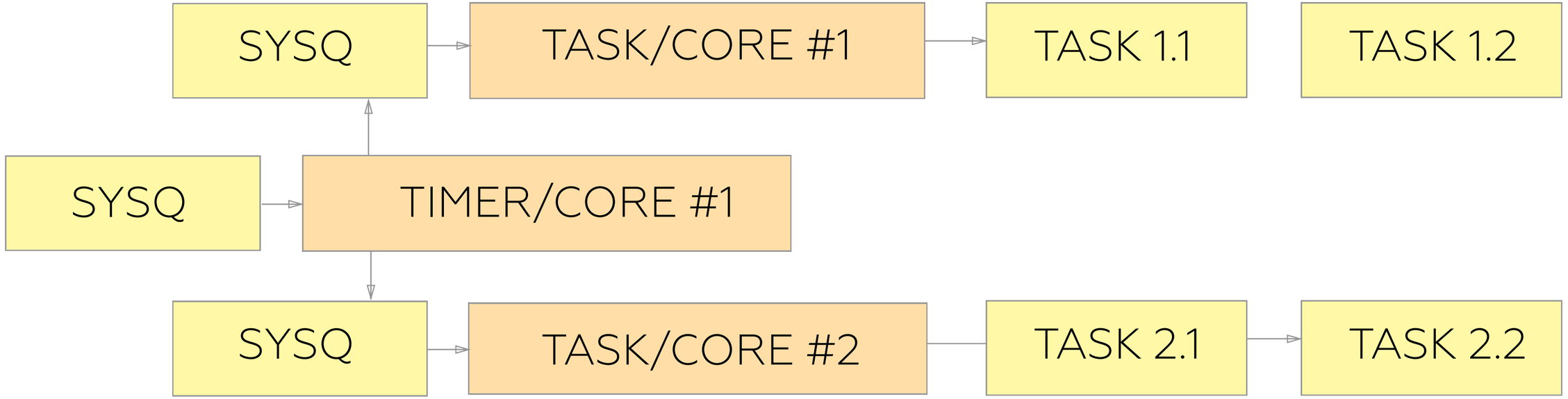
Multicursor Implementation of SUB (MPSC) for InterCore Queue Migrations and Cache Locality



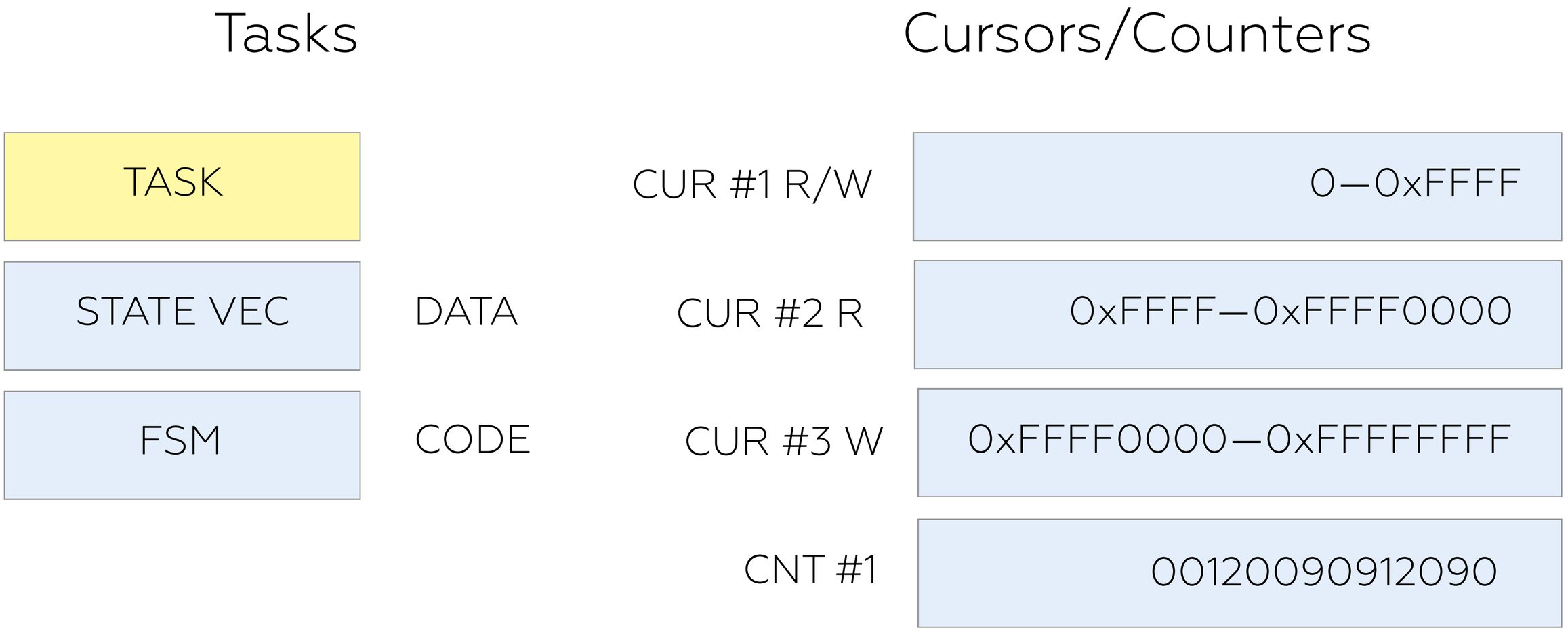


TIMERS

Scheduler Reactors can communicate throught InterCore transport for Timers.



Timer uses Linear Firing Round Robin.

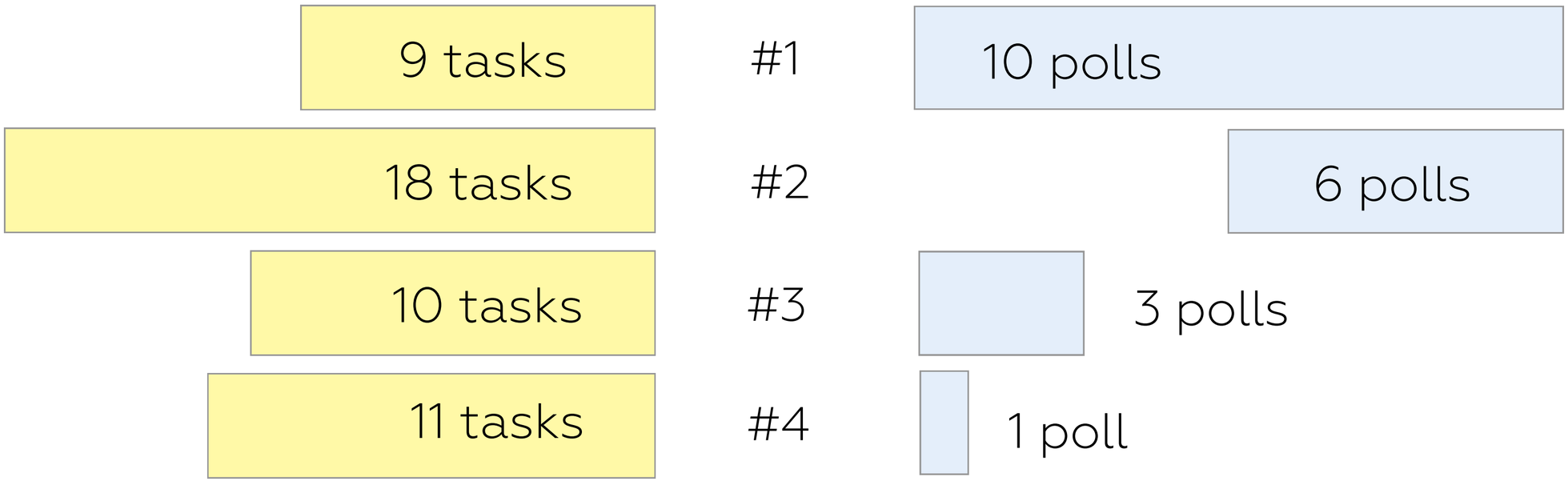

Tasks
Cursors/Counters
TASK
STATE VEC
FSM
DATA
CODE
CUR #1 R/W
0—0xFFFF
CUR #2 R
0xFFFF—0xFFFF0000
CUR #3 W
0xFFFF0000—0xFFFFFFFF
CNT #1
00120090912090

Capacity: 239 Time: 20

Workload: 48 Total: 400

Avg Task Consumtion
Accumulated in the Task Stream

$$\Sigma \text{ Tasks} * \text{Polls} * \text{AvgTime} = \text{Capacity}$$



prios: [10,6,3,1]

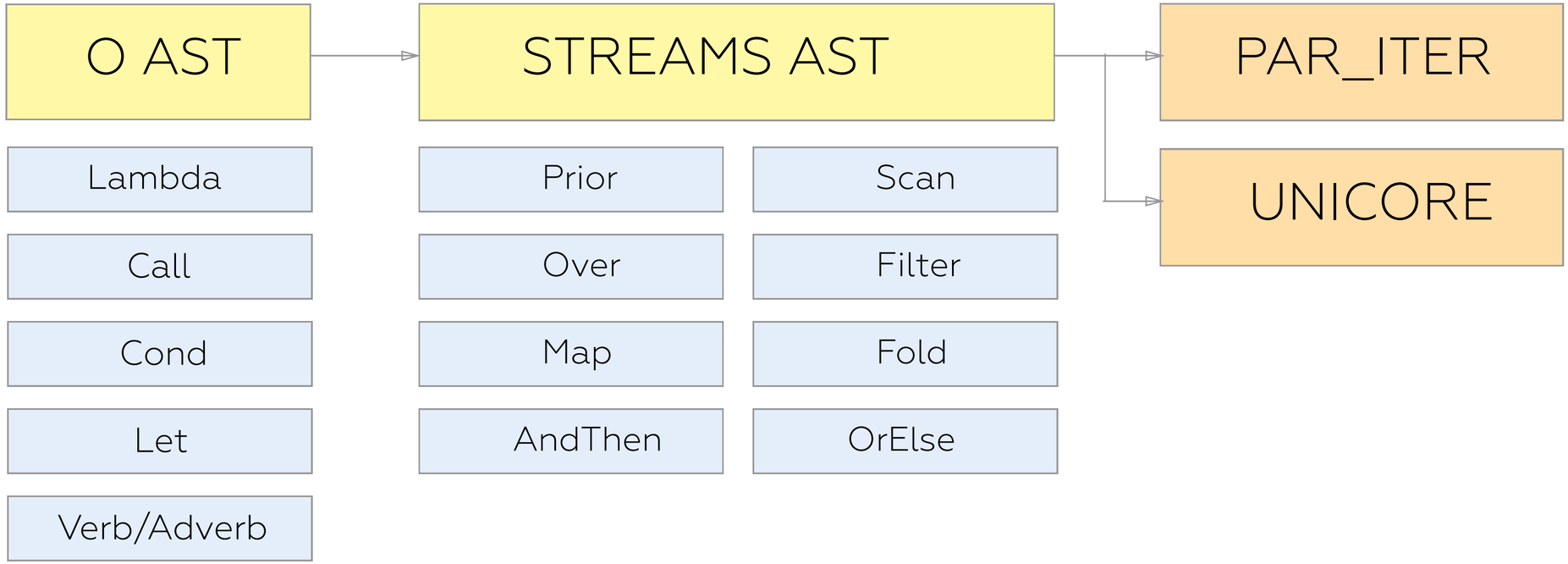
ITERATORS

```
+/{x*y}[(1;3;4;5;6);
        (2;6;2;1;3)]

+/{x*y}[vec1;vec2]

vec1.iter().zip(vec2
.iter()).map(|(i, j)|
i * j).sum()
```

```
movdqu    16(%rdx,%rax,4), %xmm2
movdqu    16(%rdi,%rax,4), %xmm3
pshufd    $245,   %xmm2,   %xmm4
pmuludq   %xmm3,  %xmm2
pshufd    $232,   %xmm2,   %xmm2
pshufd    $245,   %xmm3,   %xmm3
pmuludq   %xmm4,  %xmm3
pshufd    $232,   %xmm3,   %xmm3
punpckldq %xmm3,  %xmm2
paddd     %xmm2,  %xmm1
movdqu    (%rdx,  %rax,4), %xmm2
movdqu    (%rdi,  %rax,4), %xmm3
pshufd    $245,   %xmm2,   %xmm4
pmuludq   %xmm3,  %xmm2
pshufd    $232,   %xmm2,   %xmm2
pshufd    $245,   %xmm3,   %xmm3
pmuludq   %xmm4,  %xmm3
pshufd    $232,   %xmm3,   %xmm3
punpckldq %xmm3,  %xmm2
paddd     %xmm2,  %xmm0
```

INTERPRETER

Unified Pipeline of Language and Streams Interpretation for Unicore and Multicore

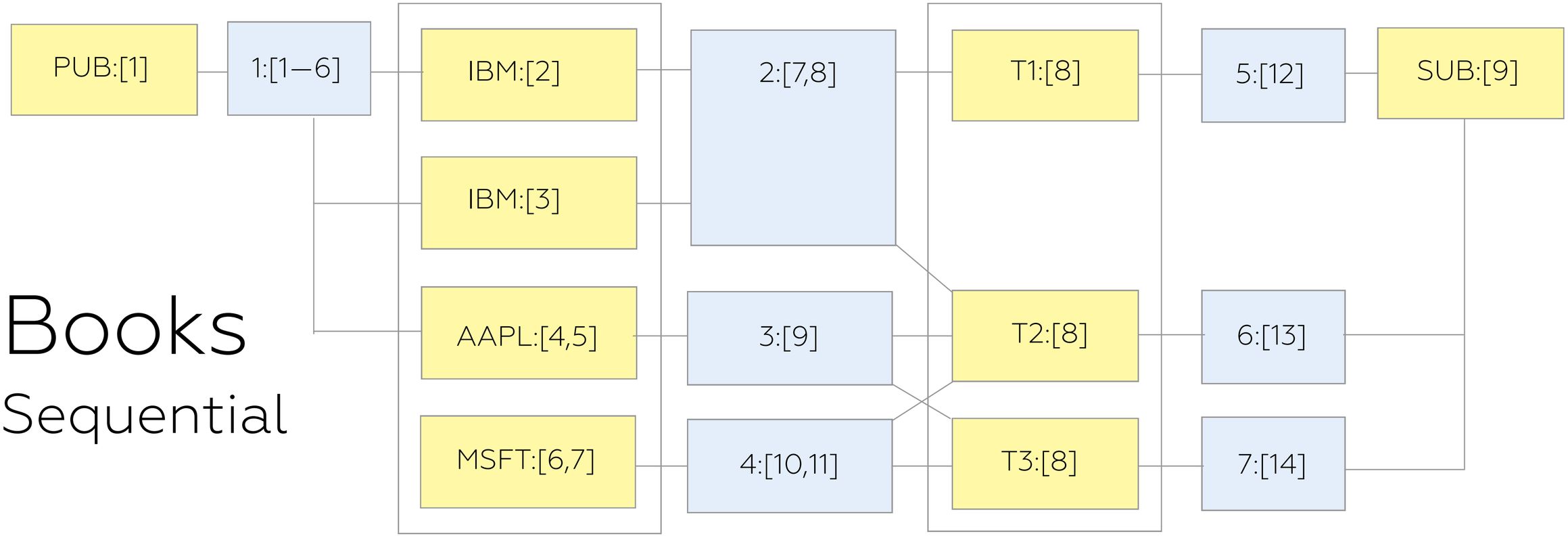


The motivation is to keep LLVM vectorizer continuous happy

```
Console is listening...
>
ring[reader;mem[0;16]];
ring[writer;mem[0;16]];
cursor[1;writer;1];
split[1;2;50];
split[2;3;50];
split[1;4;50];
cursor[5;reader;1];
split[5;6;50];
split[5;7;overlapped];
reactor[aux;0;mod[console;network]];
reactor[timercore;1;mod[timer]];
reactor[core1;2;mod[task]];
reactor[core2;3;mod[task]];
spawn[1;80;AAPL;trader1;core1];
spawn[2;80;EEM-SPY-GDX;trader1;core1];
spawn[3;20;AMI;trader1;core1];
spawn[5;80;GOOG;trader2;core2];
spawn[4;80;FB-NFLX-AMZN;trader2;core2];
timer[timer1;core1;SPY;rule1;t1;notify];
list[reactors];
list[rings];
list[cursors;writer];
list[core1];
list[timercore];
send[1;message1];
send[1;message2];
dump[1;mem[0;100]];
show[recv;1];
```

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

12x CPU Cores: In: [1] Order Books: [2,3,4,5,6,7] Traders: [8] Out: [9]

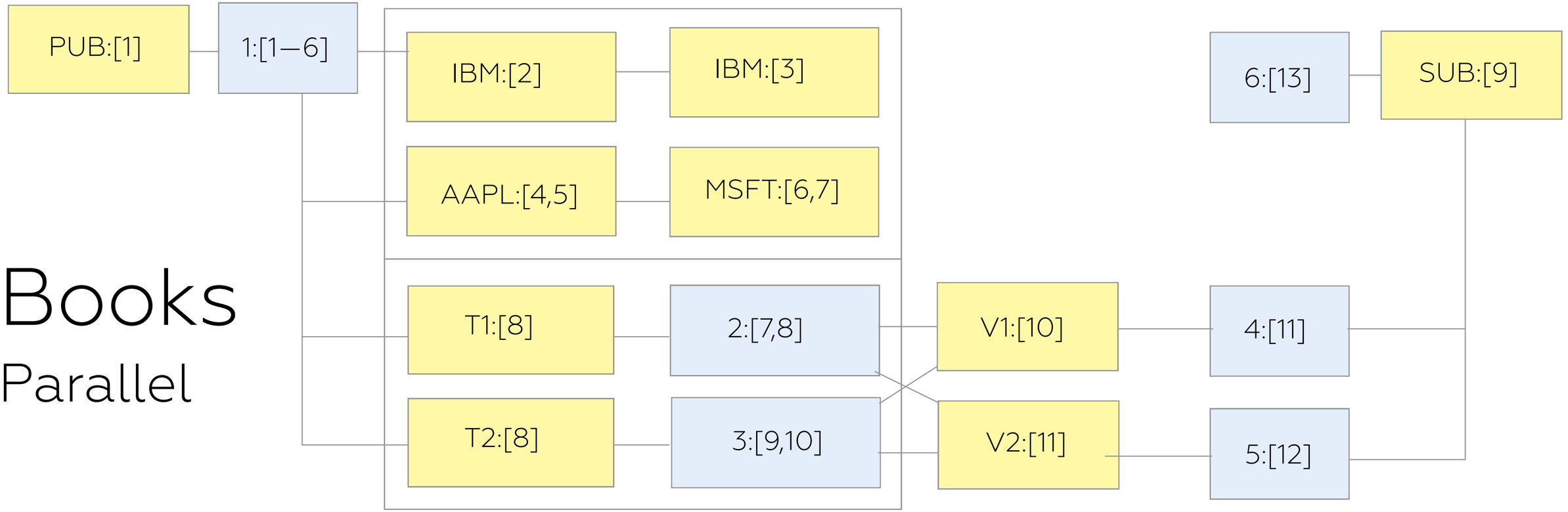


Books

Sequential

8x32K MEM Regions: Input Queue: [1] Reducing Queues: [2,3,4,5,6,7]

12x CPU Cores: In: [1] Order Books: [2,3,4,5,6,7] Traders: [8] Out: [9]
Venues: [10,11]



Books

Parallel

8x32K MEM Regions: Input Queue: [1] Reducing Queues: [2,3,4,5,6]

Book: AAPL

id	side	time	vol	price	venue
3	ASK	09:05:01:123871012	200	20.30	1
1	ASK	09:01:12:192090139	100	20.30	2
2	ASK	09:03:25:716945237	100	20.25	1
5	BID	09:08:42:134673465	200	20.20	1
4	BID	09:06:11:784316783	100	20.15	1
6	BID	09:09:37:834852874	200	20.15	2