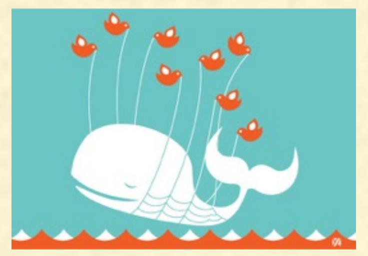
PELIKAN
"CACHE À LA CARTE"

A framework for building *production-ready* cache in datacenters.

Yao Yue, Twitter Inc
@thinkingfish

ABOUT ME

- at Twitter since 2010 
- working on cache this whole time
- worn every hat

ABOUT THE TALK

- mostly *not* about cache
- based on twitter's use of cache
- a quest for high-quality infrastructure

Context

DEPLOYMENT & SCALE

- many clusters, in containers, automated deploy
- qps: from thousands to tens of millions
- data models, query size and access pattern- all over the map

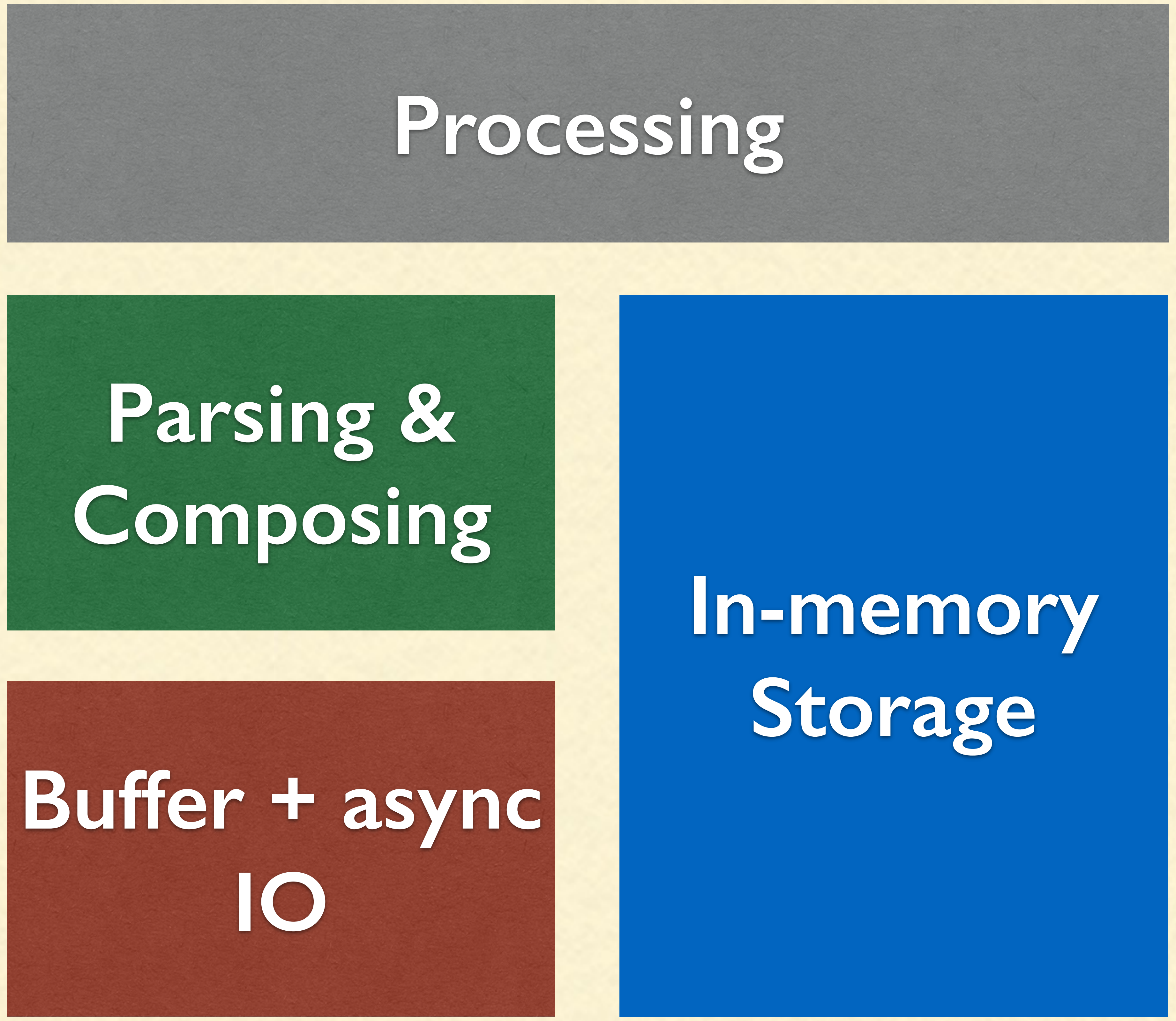
A “SOLVED” PROBLEM

Twemcache

Redis

Slimcache

Fatcache*



COMMON CHALLENGES

- many ways to fall below SLA
- hotkeys and DDoS
- hard to debug
- capacity planning surprises

THE CACHE WE WANT

- covers all our use cases
- easy and fun to work on
- is production-ready

A DIRTY LITTLE SECRET

we have little idea about what production-readiness demands

- rely on “battle-tested” solution
- “f*ck it”

PRODUCTION-READY CACHE

PREDICTABLE

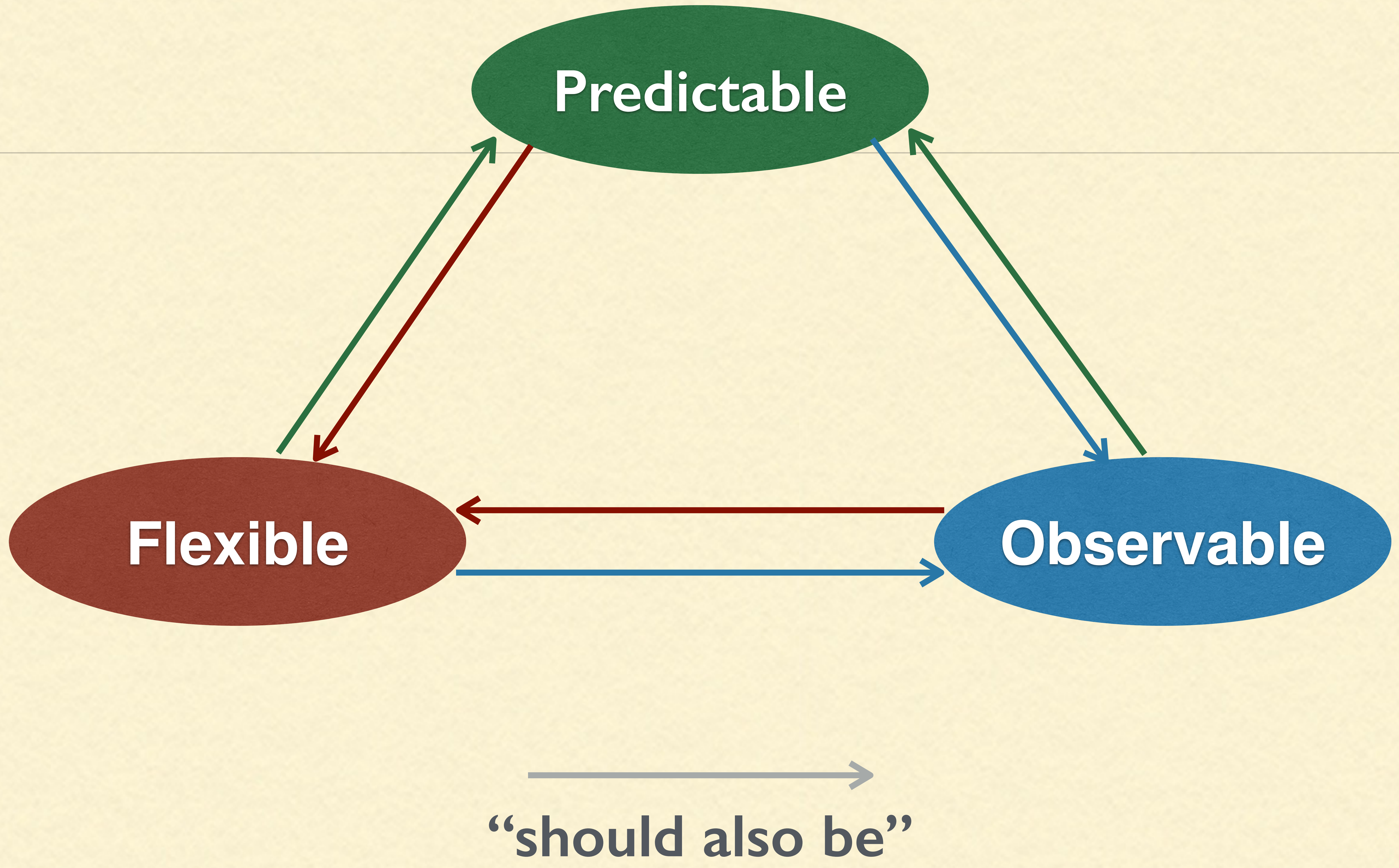
- tail-latency & performance
- failure behavior & degradation
- resource footprint

OBSERVABLE

- ready to be monitored
- debuggable
- reveals internal flow
- analytics-friendly

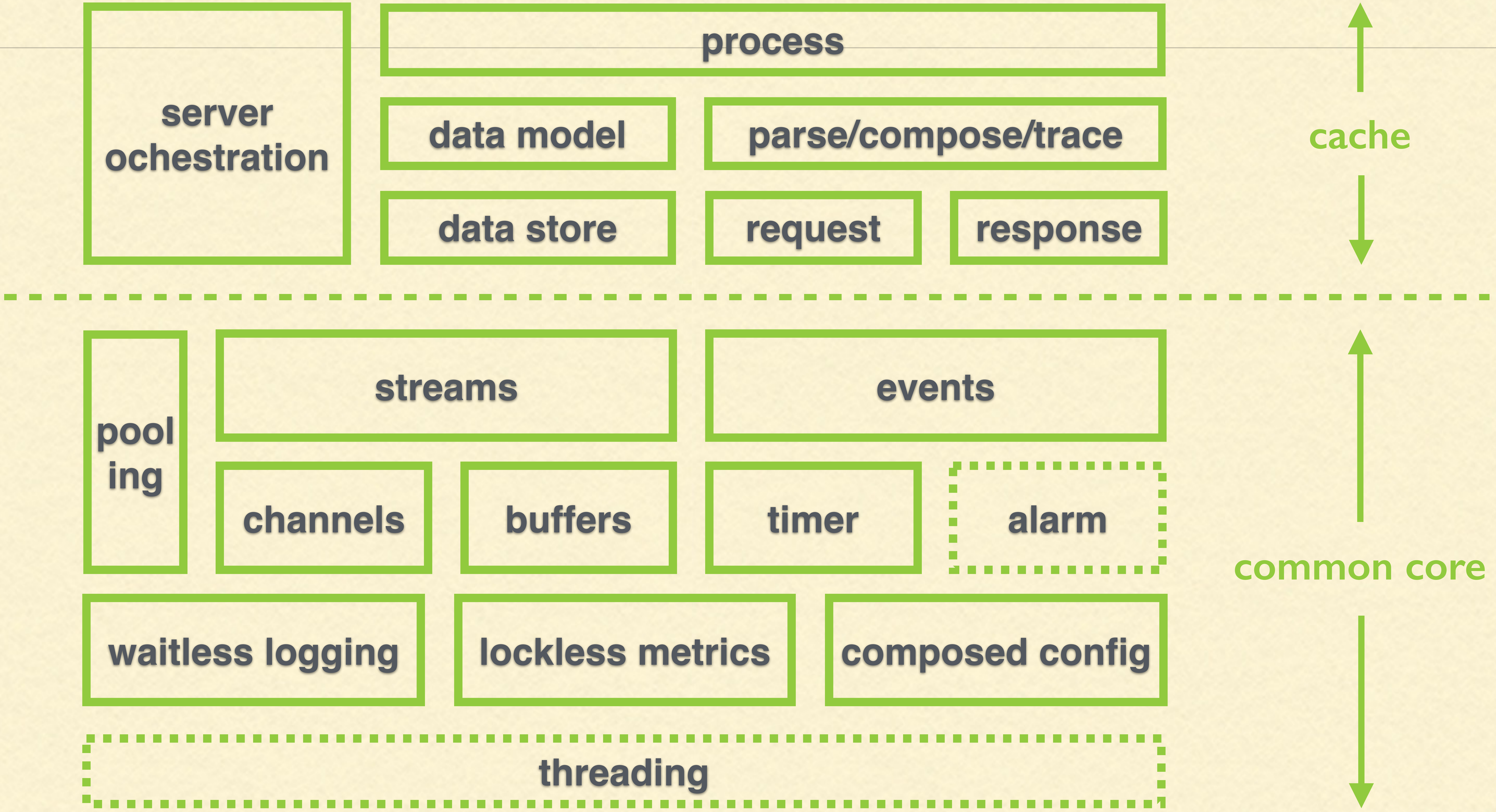
FLEXIBLE

- configurable
- composable
- quick to develop features


Predictable
Flexible
Observable
“should also be”

“SHOW US WHAT YOU GOT”

ARCHITECTURE OVERVIEW



MODULARIZE…

- minimize surface area- no leaky abstraction
- right amount of generality

…AND FOR EACH MODULE:

- “Are the performance/failure scenarios known?”
- “How do I manage the resources?”
- “How do I get visibility? What do I track?”
- “How can I configure this? What’s good defaults?”

JUDICIOUS CODE REUSE

- clean slate design
- use tested logic
- alter to fit
- ~~future-proof~~ future-compatible

CORE DECISIONS

- common core / cache split

  everything you need for a production-ready ping server + the rest

- control / data plane split

  background thread performing non-critical tasks

DEVELOPMENT

- developers: me, Kevin Yang, Sagar Vemuri
- mostly since summer 2014
- clean-slate design v.s. ~50% existing code
- binaries: pelikan_twemcache, pelikan_slimcache, pelikan_redis*
- pilot production deploy done; load test**, universal canary to come

PRODUCTION-READINESS: SOME HIGHLIGHTS

LOG, STATS, CONFIG

ubiquitous, *paradigms*

make them cheap, configurable

make them composable

waitless logging

lockless stats

modular config

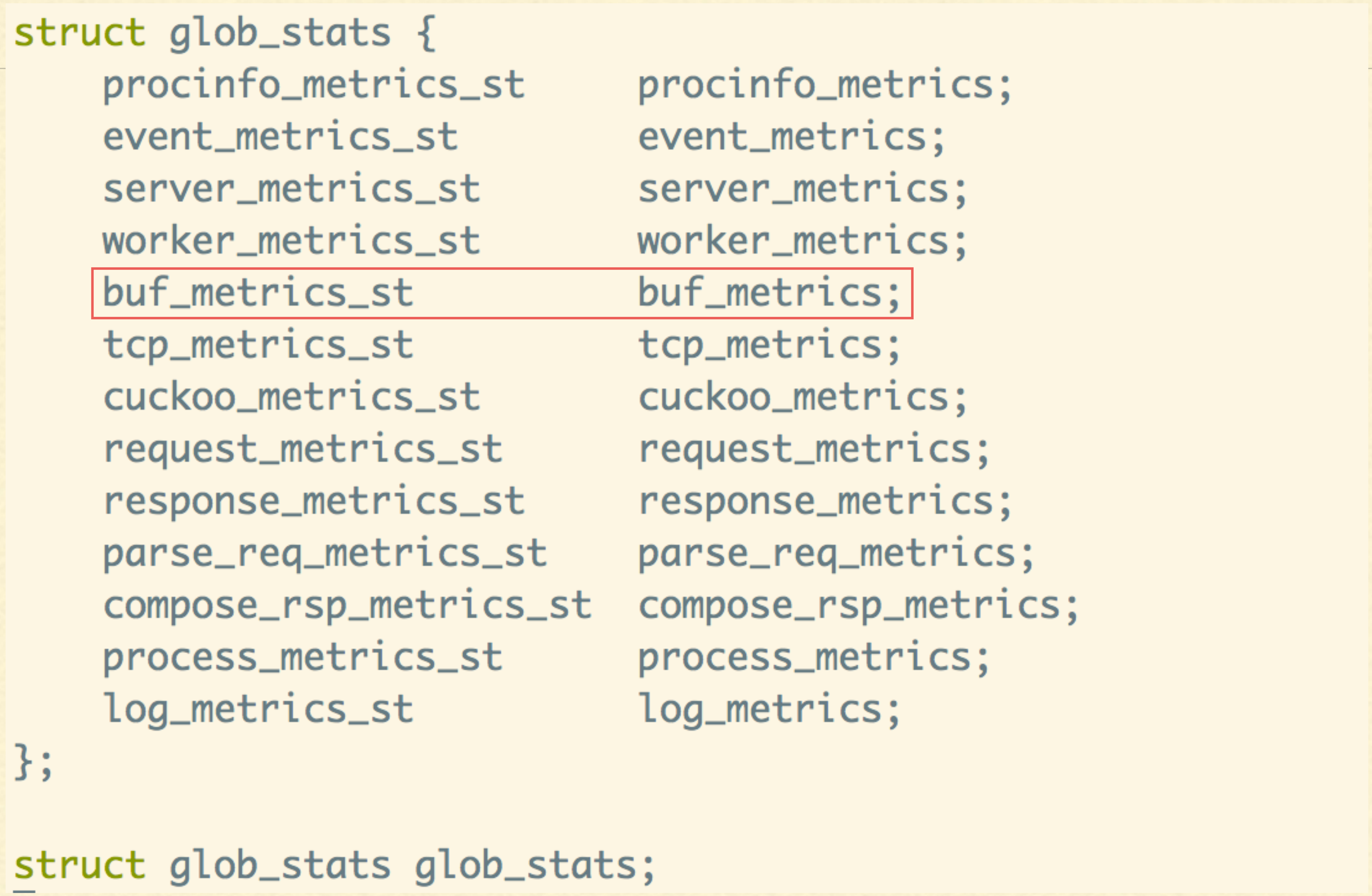
DECLARE/INITIALIZE METRICS

```
/*          name              type            description */
#define BUF_METRIC(ACTION)                                              \
    ACTION( buf_curr,         METRIC_GUAGE,   "# buf allocated"        )\
    ACTION( buf_active,       METRIC_GUAGE,   "# buf in use/borrowed"  )\
    ACTION( buf_create,       METRIC_COUNTER, "# buf creates"          )\
    ACTION( buf_create_ex,    METRIC_COUNTER, "# buf create exceptions")\
    ACTION( buf_destroy,      METRIC_COUNTER, "# buf destroys"         )\
    ACTION( buf_borrow,       METRIC_COUNTER, "# buf borrows"          )\
    ACTION( buf_borrow_ex,    METRIC_COUNTER, "# buf borrow exceptions")\
    ACTION( buf_return,       METRIC_COUNTER, "# buf returns"          )\
    ACTION( buf_memory,       METRIC_GAUGE,   "memory allocated to buf")

typedef struct {
    BUF_METRIC(METRIC_DECLARE)
} buf_metrics_st;

#define BUF_METRIC_INIT(_metrics) do {                          \
    *(_metrics) = (buf_metrics_st) { BUF_METRIC(METRIC_INIT) }; \
} while(0)
```

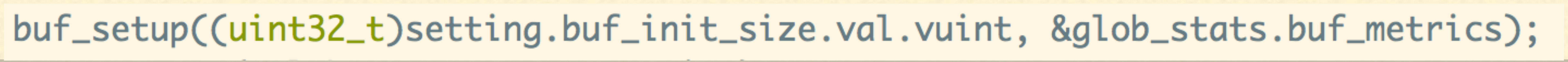
UPDATE METRICS

```
INCR(buf_metrics, buf_create);
INCR(buf_metrics, buf_curr);
INCR_N(buf_metrics, buf_memory, buf_init_size);
```

INCLUDE METRICS



```
struct glob_stats {
    procinfo_metrics_st    procinfo_metrics;
    event_metrics_st       event_metrics;
    server_metrics_st      server_metrics;
    worker_metrics_st      worker_metrics;
    buf_metrics_st         buf_metrics;
    tcp_metrics_st         tcp_metrics;
    cuckoo_metrics_st      cuckoo_metrics;
    request_metrics_st     request_metrics;
    response_metrics_st    response_metrics;
    parse_req_metrics_st   parse_req_metrics;
    compose_rsp_metrics_st compose_rsp_metrics;
    process_metrics_st     process_metrics;
    log_metrics_st         log_metrics;
};

struct glob_stats glob_stats;
```



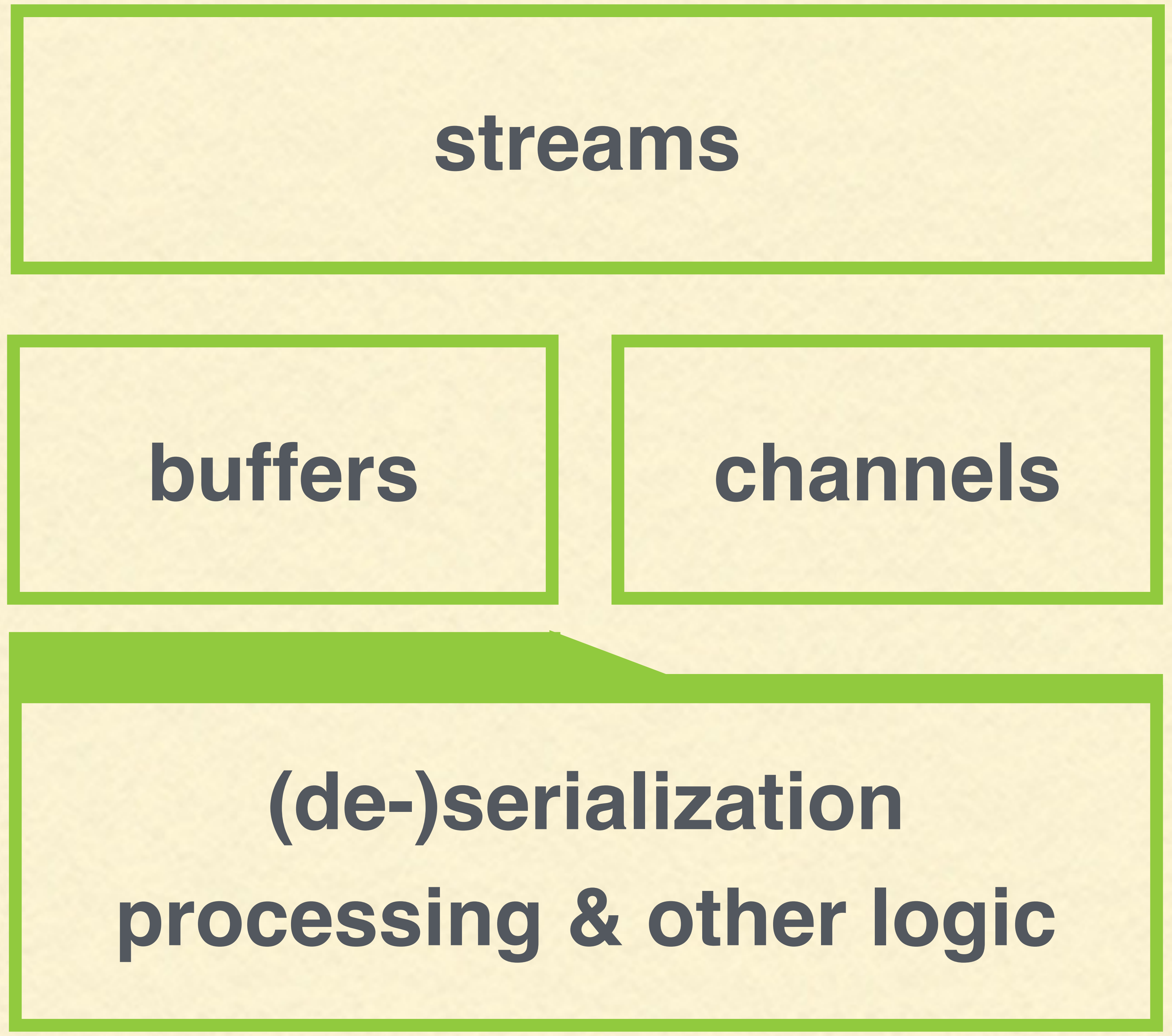
```
buf_setup((uint32_t)setting.buf_init_size.val.vuint, &glob_stats.buf_metrics);
```

BUFFER, CHANNEL, STREAM

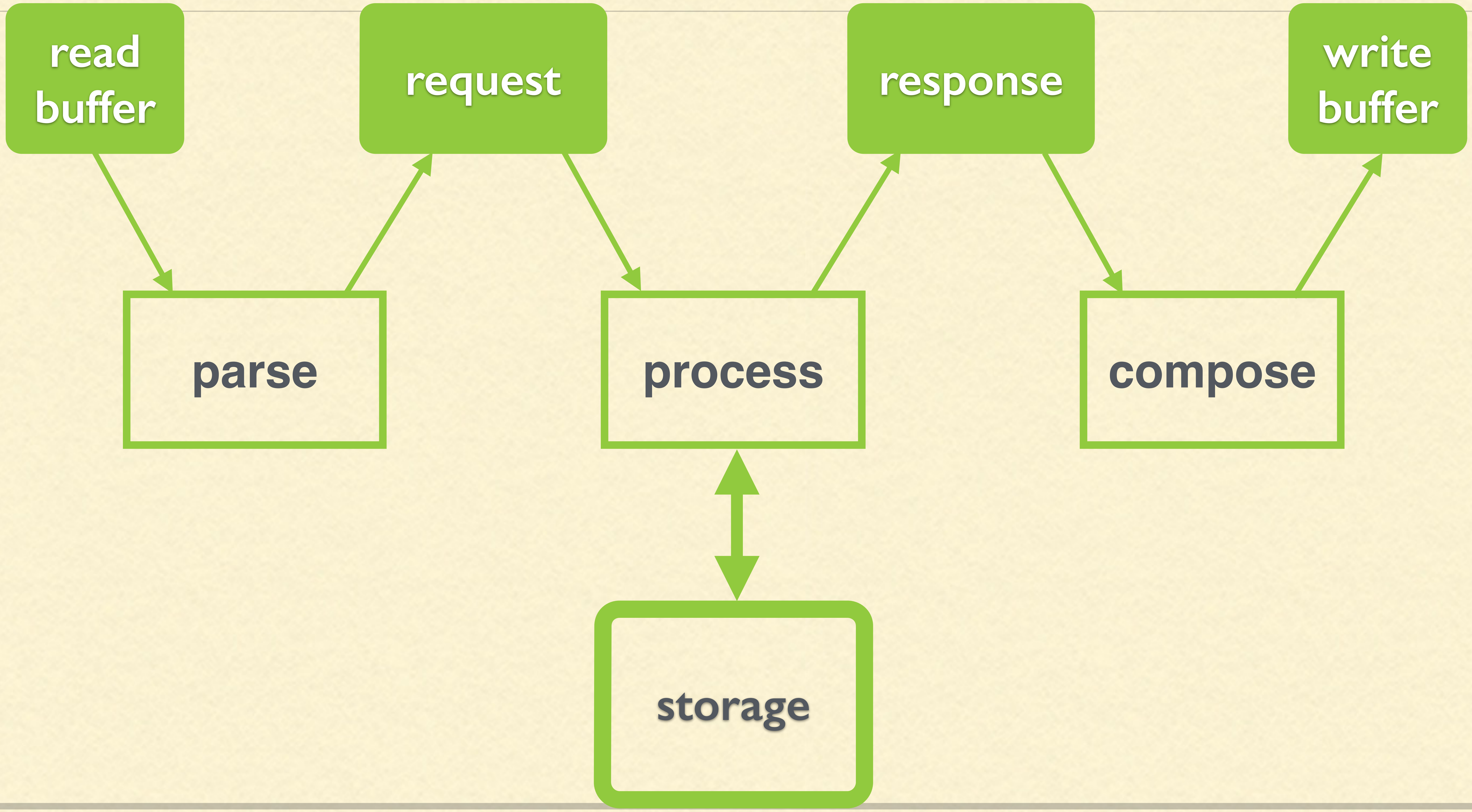
buffer connects sync/async processing

interface hides multiple implementation

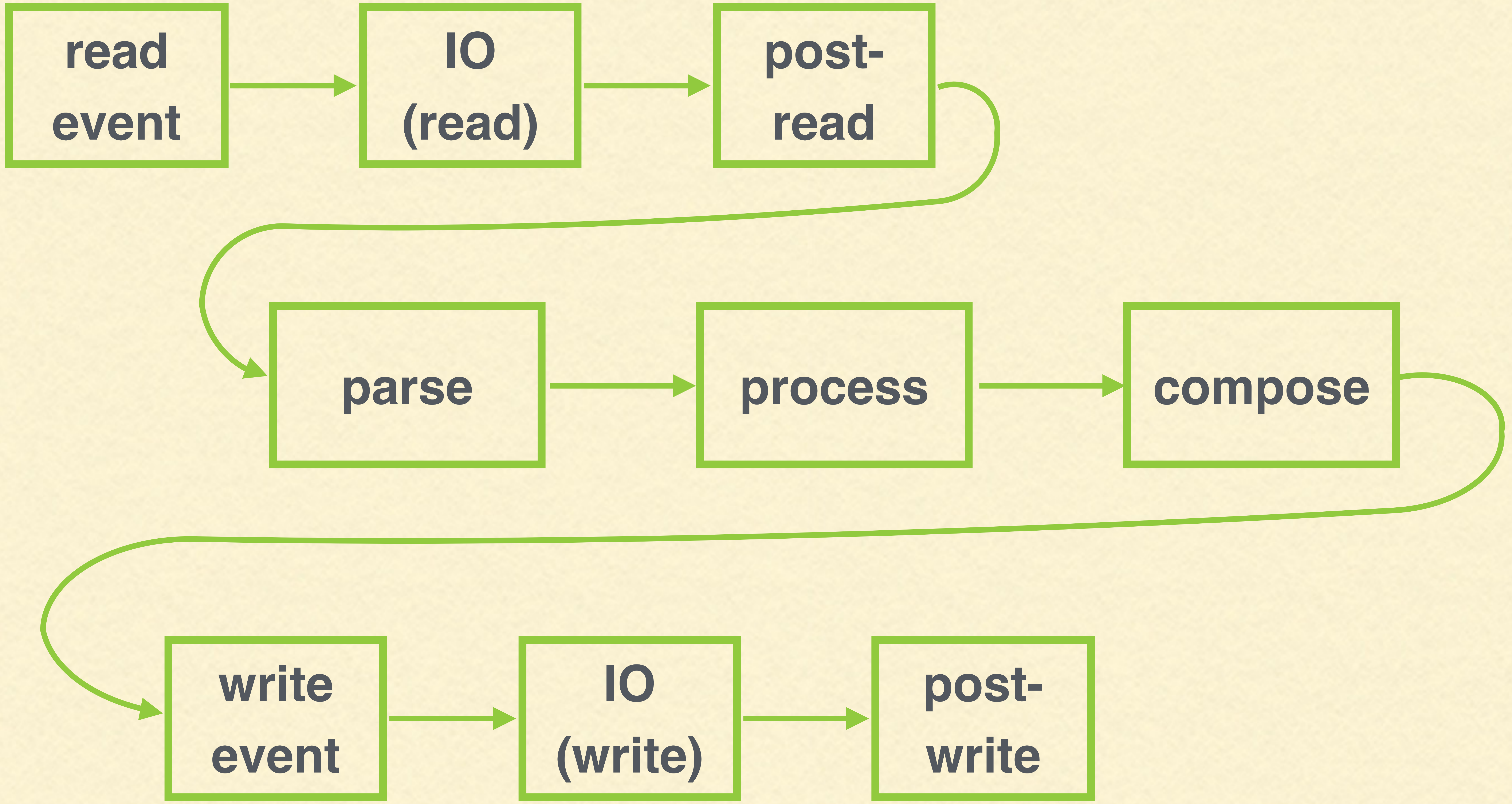
all resources can pooled, capped



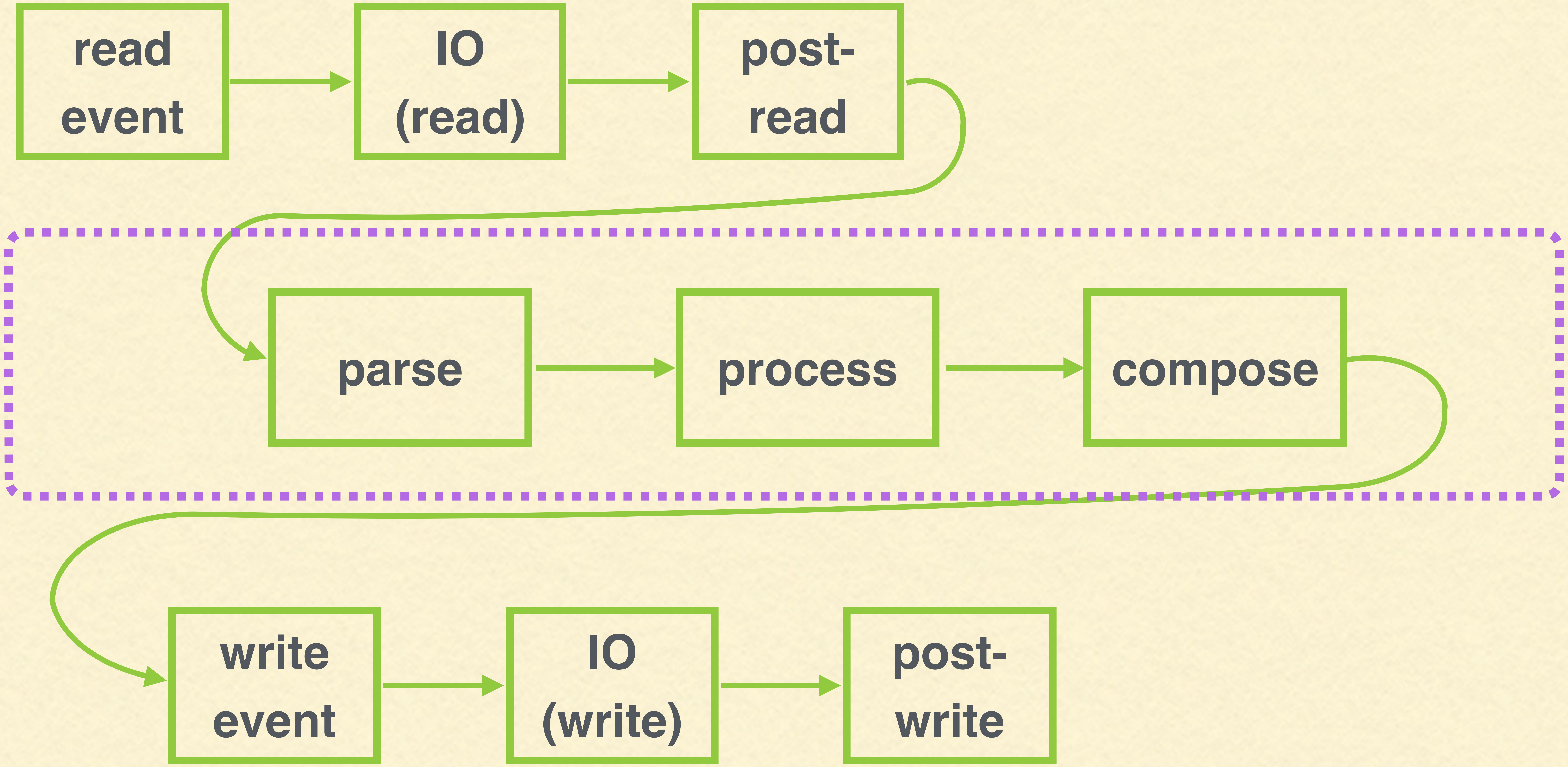
STRUCTURAL SYMMETRY



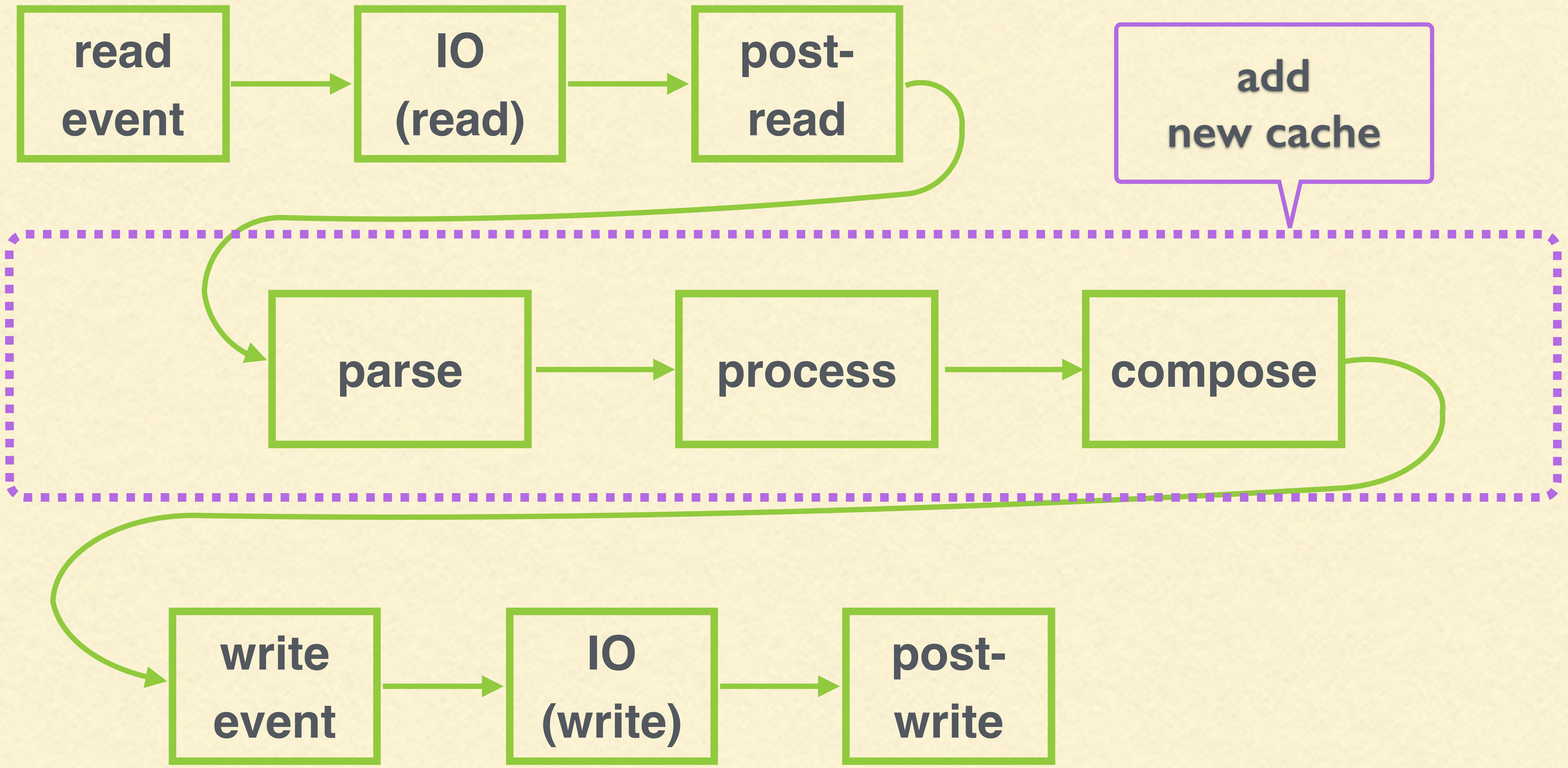
STANDARD FLOW



STANDARD FLOW



STANDARD FLOW



ADD A NEW CACHE

```
parse_rstatus_t parse_req(struct request *req, struct buf *buf);

parse_rstatus_t parse_rsp(struct response *rsp, struct buf *buf);
```

```
int compose_req(struct buf **buf, struct request *req);

int compose_rsp(struct buf **buf, struct response *rsp);
```

```
void process_request(struct response *rsp, struct request *req);
```

CASE STUDY: HOW MUCH CODE

- twemcache: 14k LOC

  uses libevent, not counted

- pelikan_twemcache: 16K LOC

  common core: 9K LOC

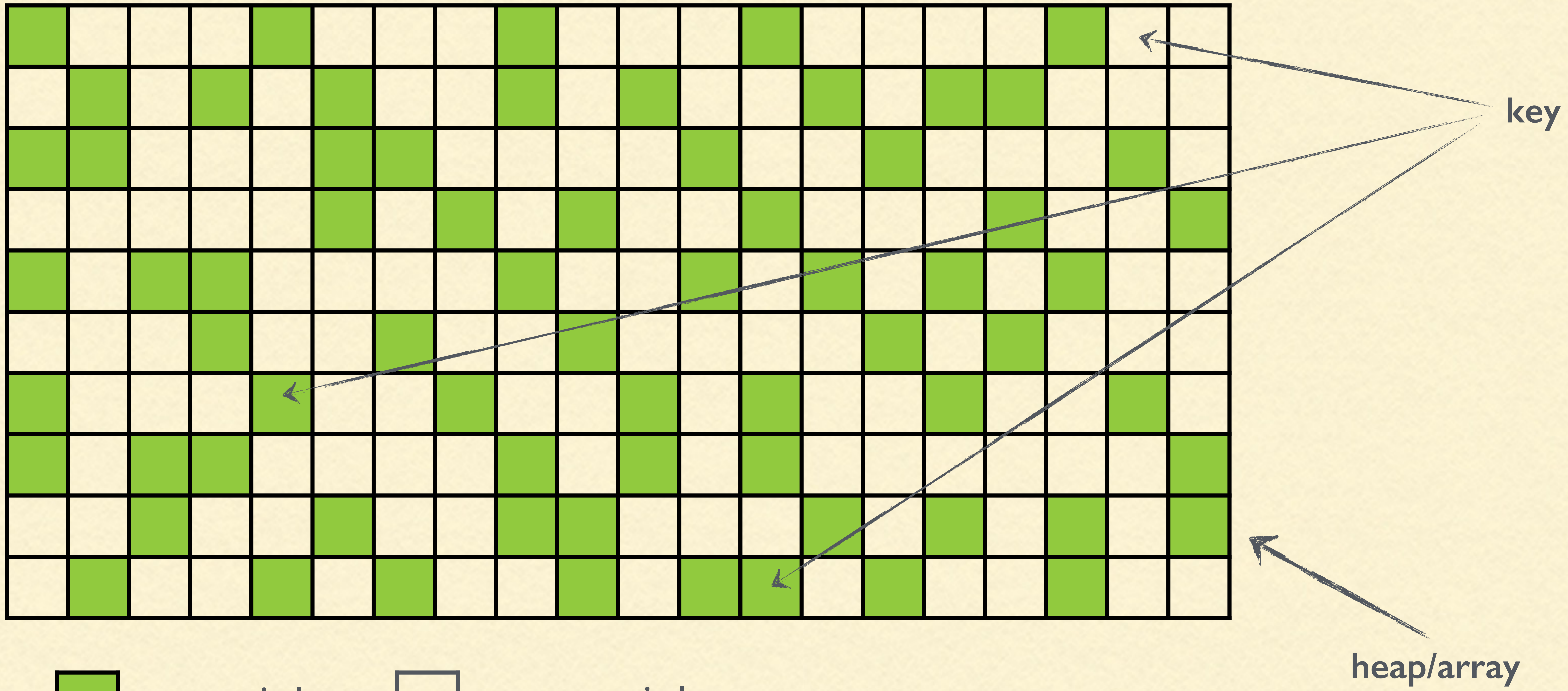
  twemcache: 7K LOC

CASE STUDY: ADD SLIMCACHE

- goal: 6B metadata instead of ~60B per key for small objects

- total: ~1600 additional LOC
- cuckoo hashing for storage: 752 LOC
- process module: 506 LOC
- other code for a new executable: 323 LOC

SLIMCACHE: CUCKOO HASHING



🟩 occupied ⬜ unoccupied

IN SUMMARY

- we spent months to achieve things we “already have”…
- … with more predictable resource, better logging/metrics…
- … out of a framework we can own long-term…
- … easy to add stuff to- the fun part!

NOW WHAT?

FOR TWITTER

- drop-in replacement for all in-house backend
- new protocol and/or new features*
- “unified cache”: build & migrate (& profit)

FOR THE REST

- we are open-sourcing! public in 2-3 weeks @pelikan_cache
- serious about OSS, develop-in-the-open kind of seriousness

- what is on your wishlist for cache?

LESSONS LEARNED

- funding this type of project was hard
- refactoring is a continual process- nothing is sacred
- form *influences* function
- forward-thinking, but be prepared for predictions to be wrong
- consistency is key to style