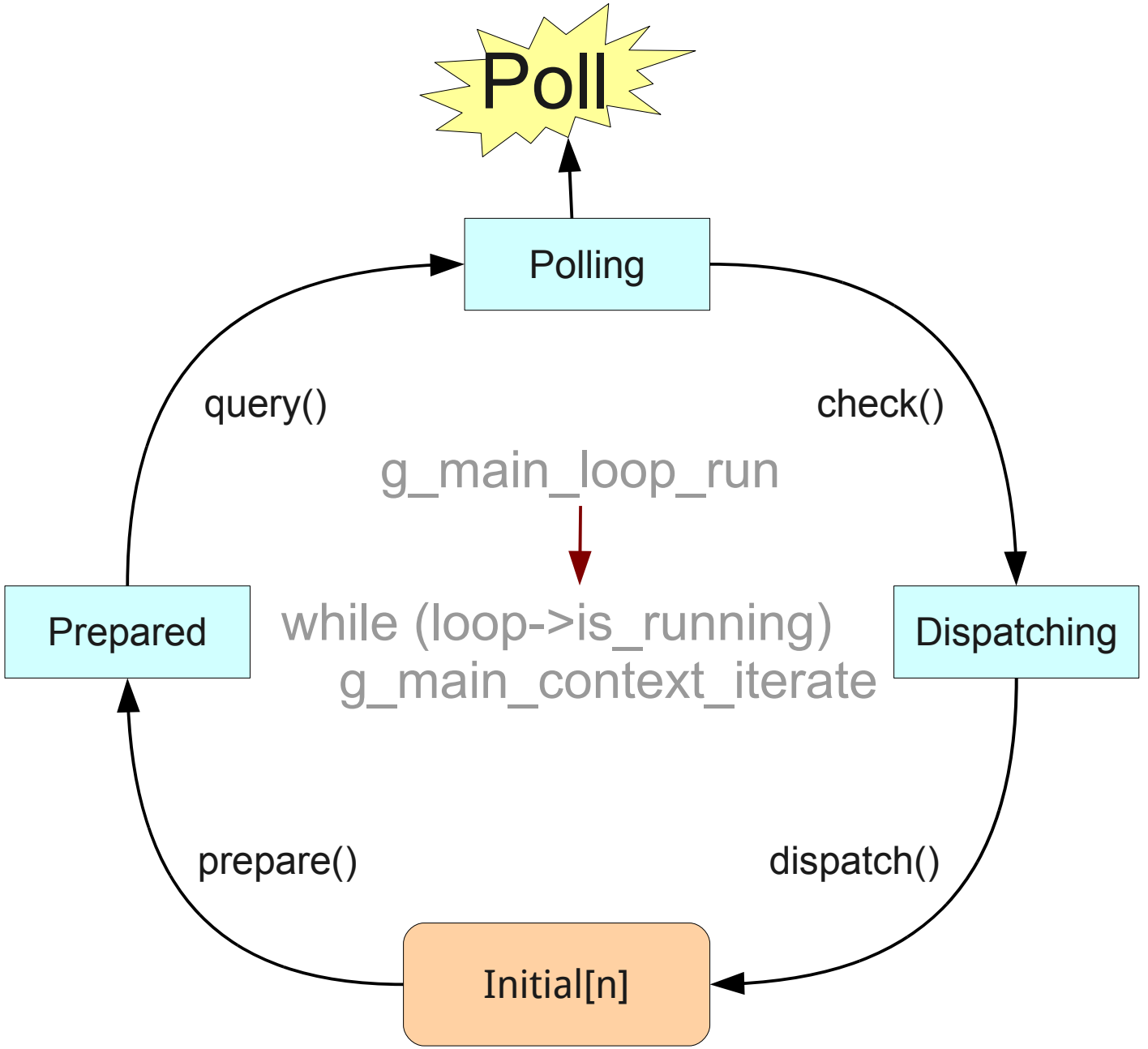
# libdispatch - event handling

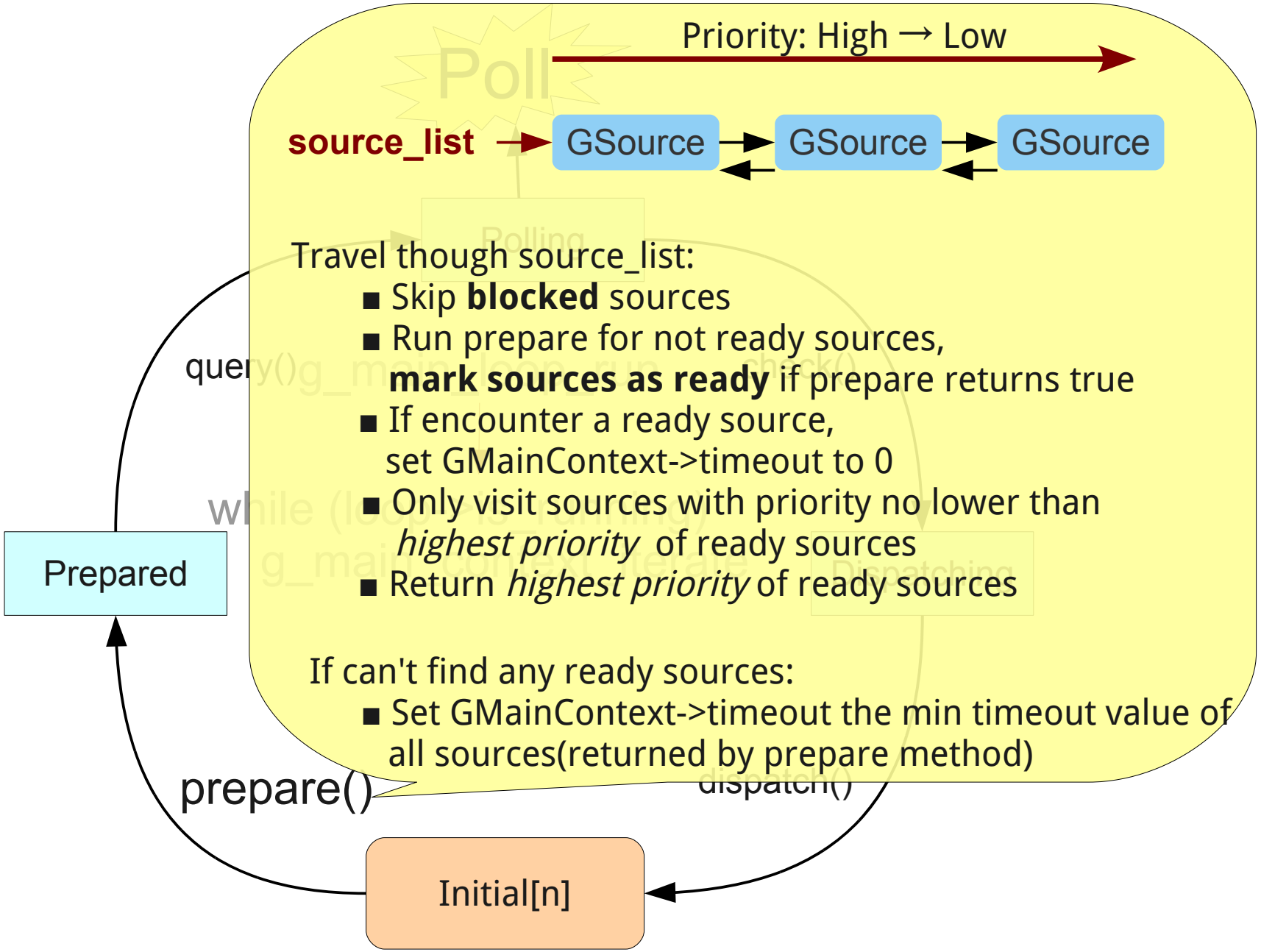
- Grand Central Dispatch
- Asynchronous & concurrent programming model
- From apple
- http://libdispatch.macosforge.org/

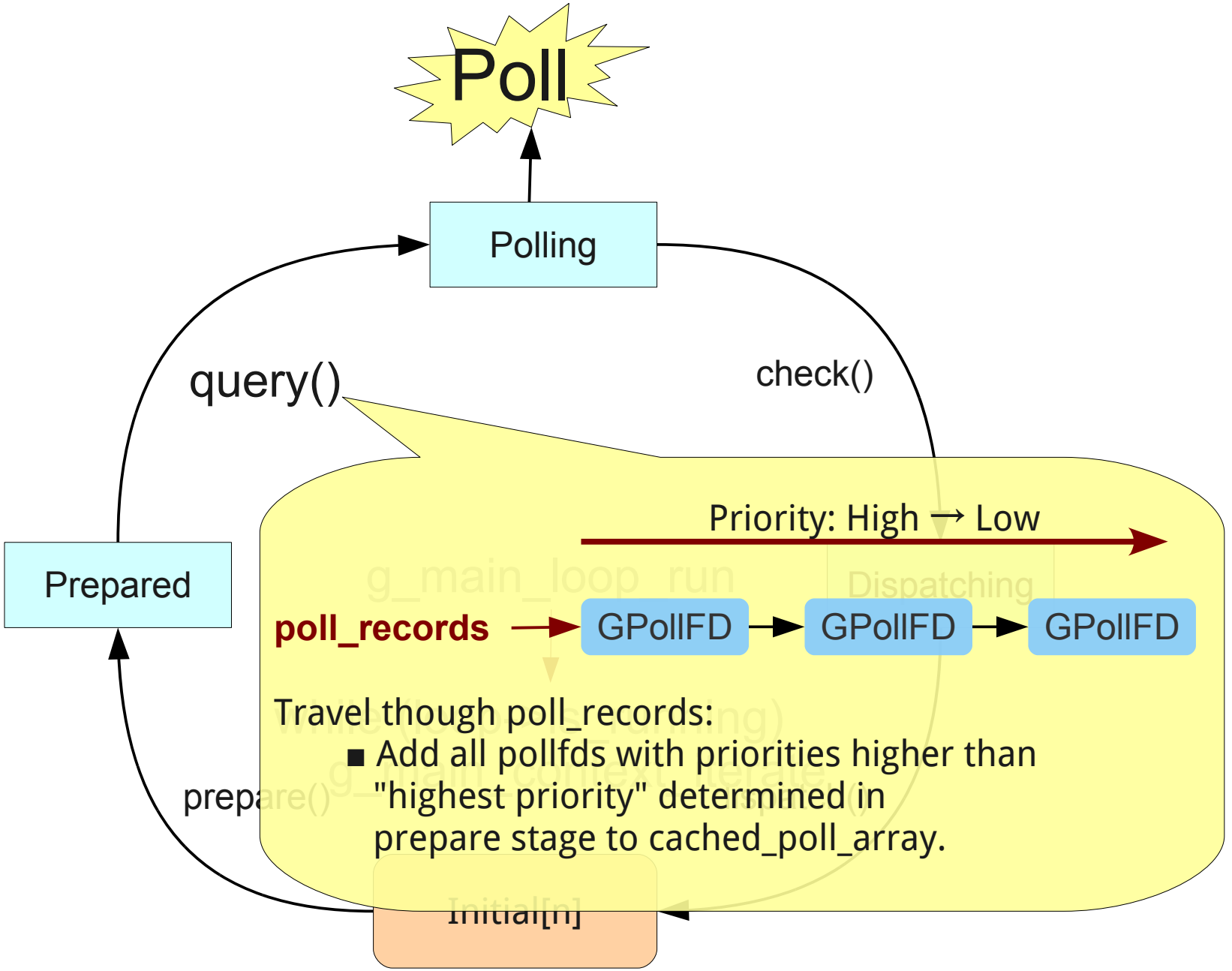
chenj@lemote.com

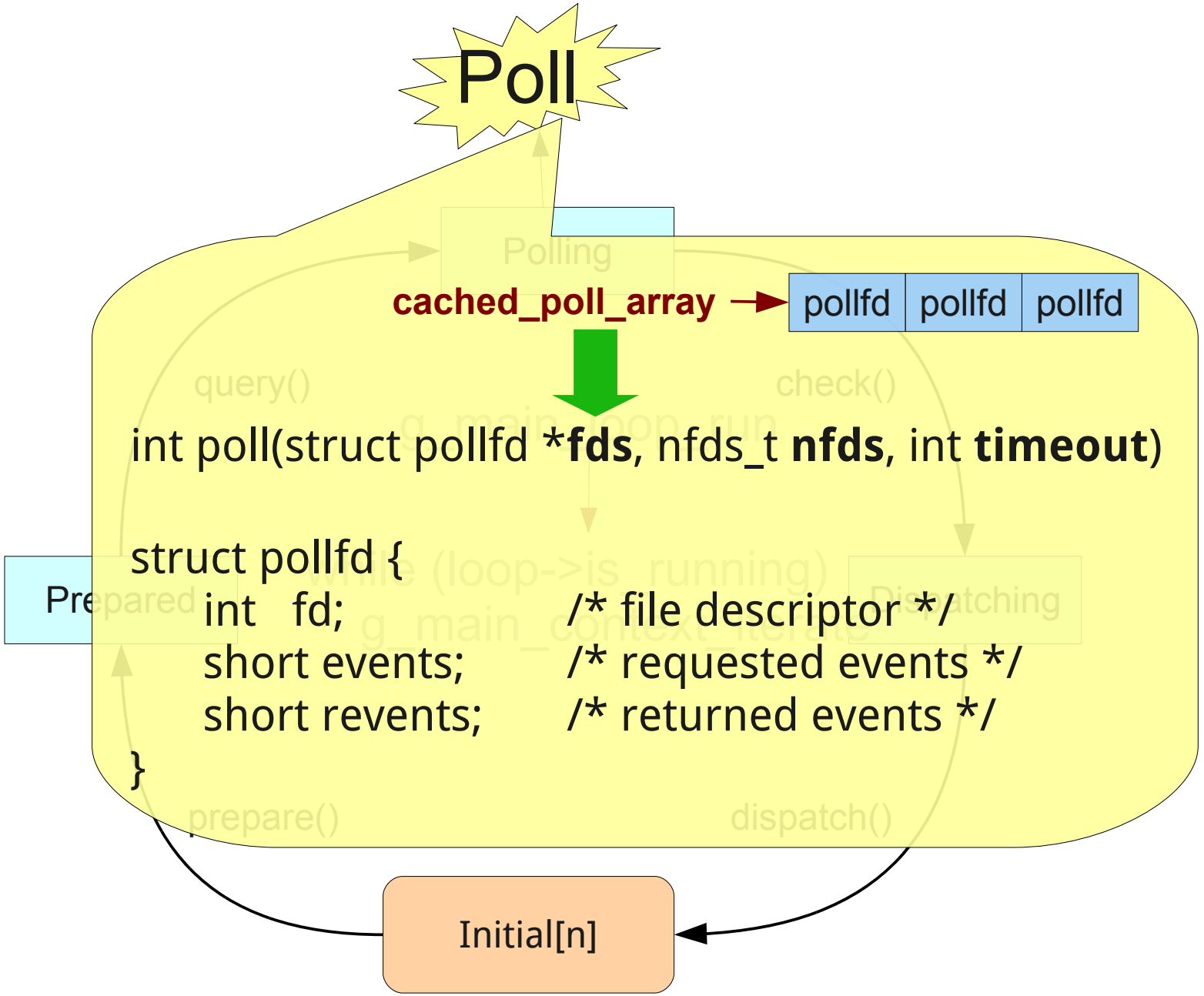
# What is event handling?

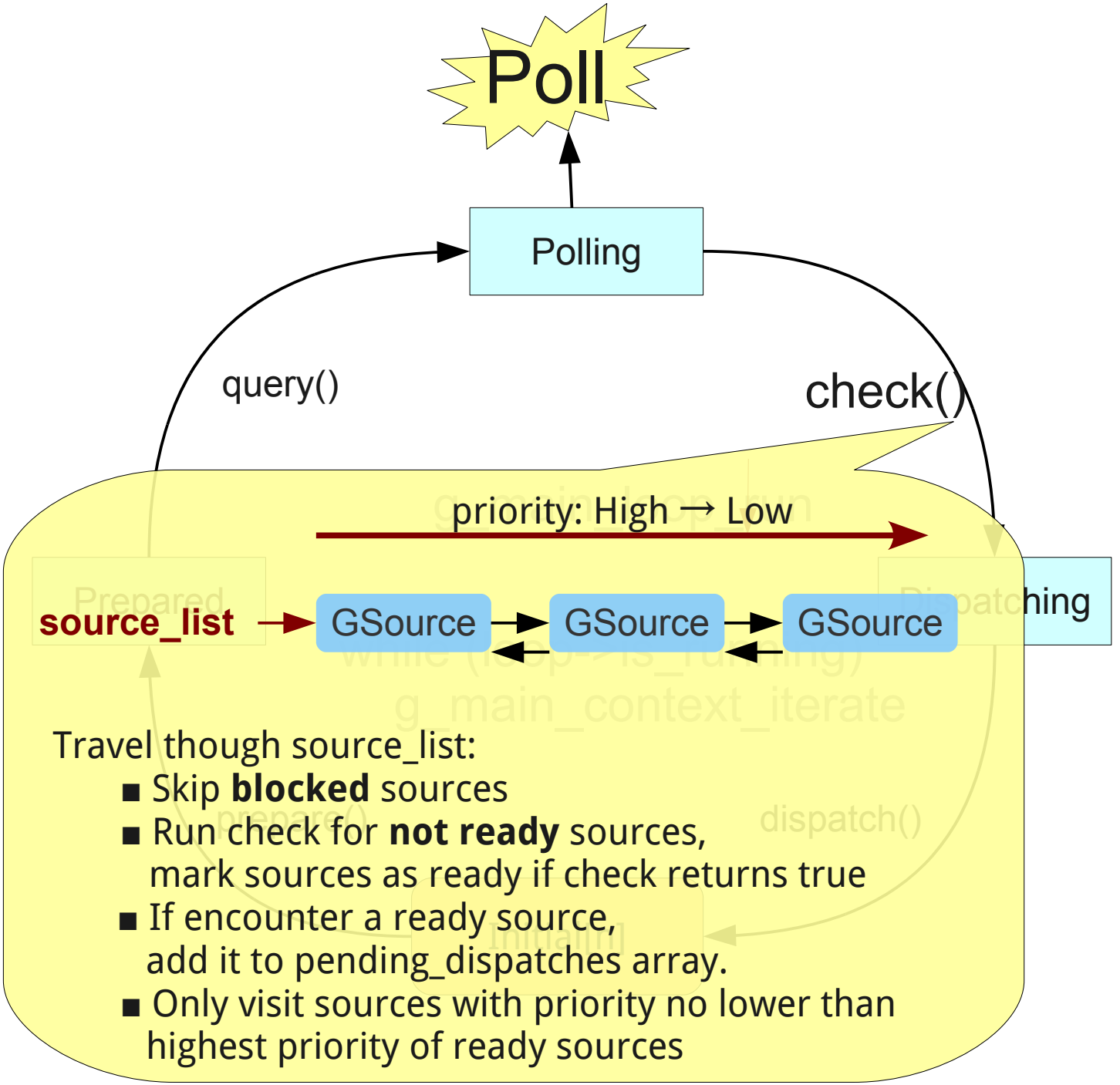
- Example: event handling in glib
  - Create a GMainContext
  - Attach one or more GSource
    - GSource: wrap event and callback
      - GSource:pollfd → **1:n**
    - Built-in GSource:
      - timeout source
      - child watch source
      - idle source
      - ...
  - Create a GMainLoop associating with GMainContext, then "g_main_loop_run"

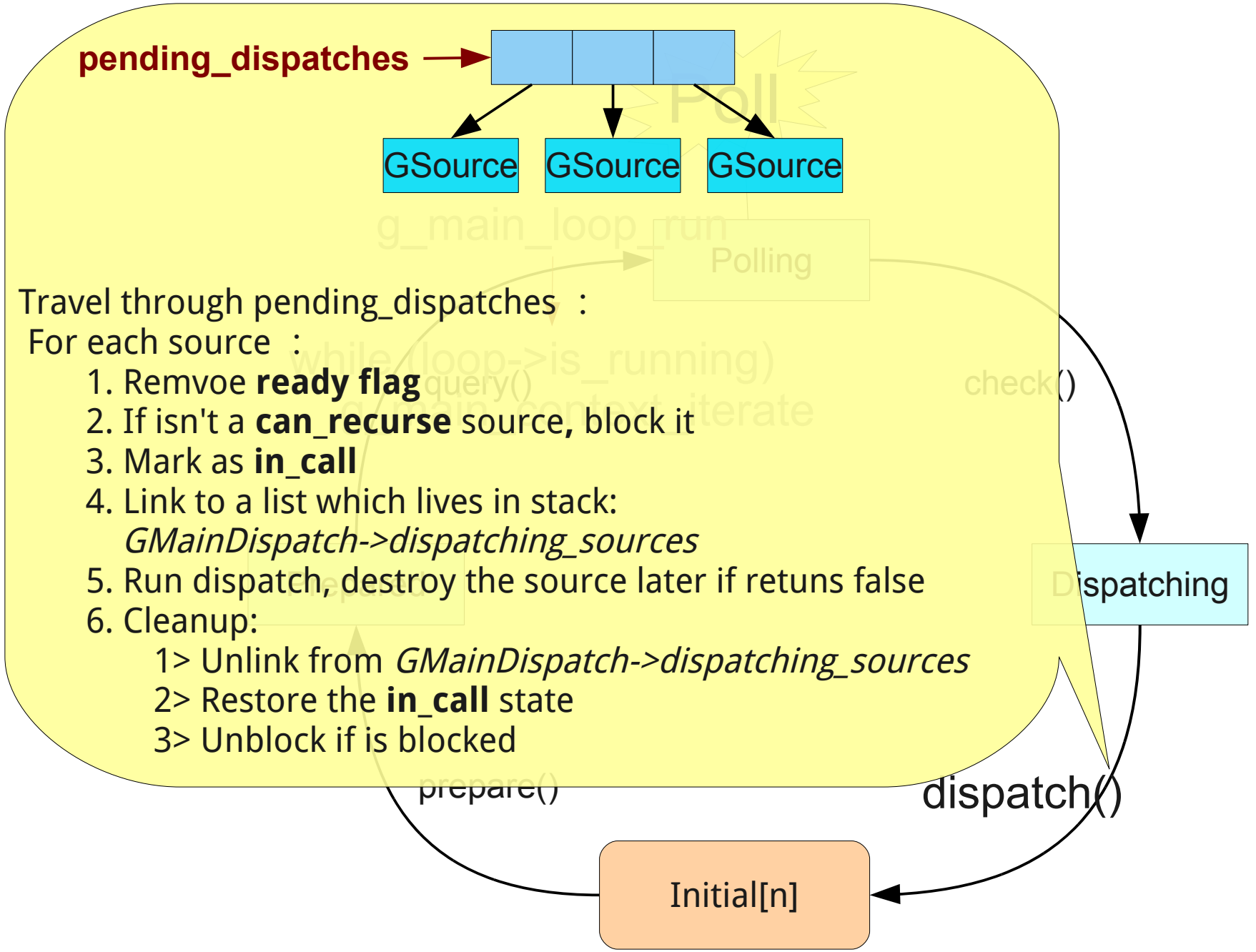

Poll
Polling
query()
check()
g_main_loop_run
while (loop->is_running)
g_main_context_iterate
Prepared
Dispatching
prepare()
dispatch()
Initial[n]


Poll
Priority: High → Low
source_list
GSource
GSource
GSource
Travel though source_list:
■ Skip **blocked** sources
■ Run prepare for not ready sources,
**mark sources as ready** if prepare returns true
■ If encounter a ready source,
set GMainContext->timeout to 0
■ Only visit sources with priority no lower than
*highest priority* of ready sources
■ Return *highest priority* of ready sources
If can't find any ready sources:
■ Set GMainContext->timeout the min timeout value of
all sources(returned by prepare method)
Prepared
query()
prepare()
dispatch()
Initial[n]


Poll
Polling
query()
check()
Prepared
Priority: High → Low
poll_records
GPollFD
GPollFD
GPollFD
Travel though poll_records:
■ Add all pollfds with priorities higher than "highest priority" determined in prepare stage to cached_poll_array.
prepare()
Initial[n]


Poll
Polling
cached_poll_array
pollfd
pollfd
pollfd
query()
check()
int poll(struct pollfd *fds, nfds_t nfds, int timeout)
struct pollfd {
int fd; /* file descriptor */
short events; /* requested events */
short revents; /* returned events */
}
Prepared
Dispatching
prepare()
dispatch()
Initial[n]


Poll
Polling
query()
check()
priority: High → Low
source_list
GSource
GSource
GSource
Travel though source_list:
■ Skip **blocked** sources
■ Run check for **not ready** sources,
mark sources as ready if check returns true
■ If encounter a ready source,
add it to pending_dispatches array.
■ Only visit sources with priority no lower than
highest priority of ready sources


pending_dispatches
GSource
GSource
GSource
Travel through pending_dispatches :
For each source :
1. Remvoe ready flag
2. If isn't a can_recurse source, block it
3. Mark as in_call
4. Link to a list which lives in stack:
GMainDispatch->dispatching_sources
5. Run dispatch, destroy the source later if retuns false
6. Cleanup:
1> Unlink from GMainDispatch->dispatching_sources
2> Restore the in_call state
3> Unblock if is blocked
Dispatching
dispatch()
Initial[n]

# Event handling in glib

- Call *g_main_context_iteration* or *g_main_loop_run in callback* in callback:
  - **can_recurse** GSource
- Running callback will slow down event handling – Long running callback leads to bad responsiveness

# Event handling in libdispatch

- dispatch_source_t: wap event and callback
  - source:kevent → n:1
  - Set a event callback and (optional) cancel callback
  - Set source's target queue – callback runs in target queue
  - Source's priority is determined by its target queue
- Monitor event and do early process in *mgr queue*
  - In a single thread of the highest priority thread pool
  - Based on kevent
  - No priority in monitoring and early processing stage

# Event handling: libdispatch vs glib

- libdispatch has better responsiveness
  - Running callback will not slow down event handling
  - kevent is more efficient than poll
- glib can run multi-GMainContexts in multi-threads

# Create/Setup source

- dispatch_source_create
  - dispatch_source_t inherits from dispatch_queue_t
  - Specify source's type – DISPATCH_SOURCE_TYPE_DATA_ADD、DISPATCH_SOURCE_TYPE_DATA_OR、 DISPATCH_SOURCE_TYPE_MACH_RECV、 DISPATCH_SOURCE_TYPE_MACH_SEND、 DISPATCH_SOURCE_TYPE_PROC、 DISPATCH_SOURCE_TYPE_READ、 DISPATCH_SOURCE_TYPE_SIGNAL、 DISPATCH_SOURCE_TYPE_TIMER、 DISPATCH_SOURCE_TYPE_VNODE、 DISPATCH_SOURCE_TYPE_WRITE
  - Created in SUSPEND state
- dispatch_source_set_event_handler_f

# Run source(Installation stage1)

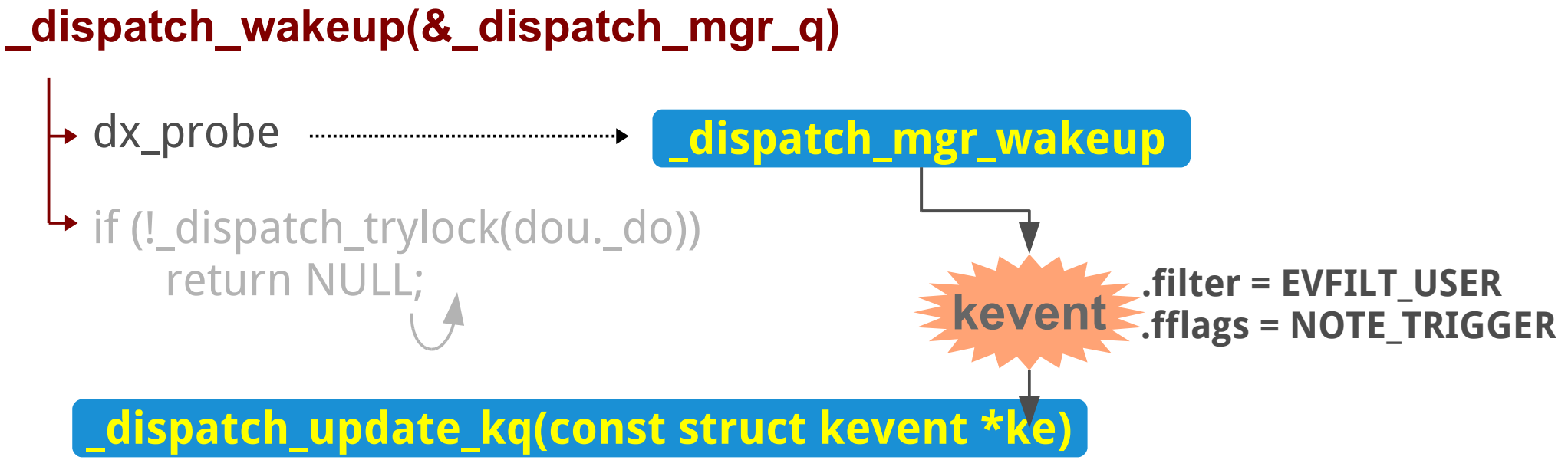
- dispatch_resume
  - _dispatch_wakeup
    - send source to its target queue
- Invoke source in target queue (_dispatch_queue_invoke)
  - _dispatch_source_invoke → redirect to *mgr queue*
- Wake up *mgr queue* (_dispatch_wakeup)

# Wake up mgr queue

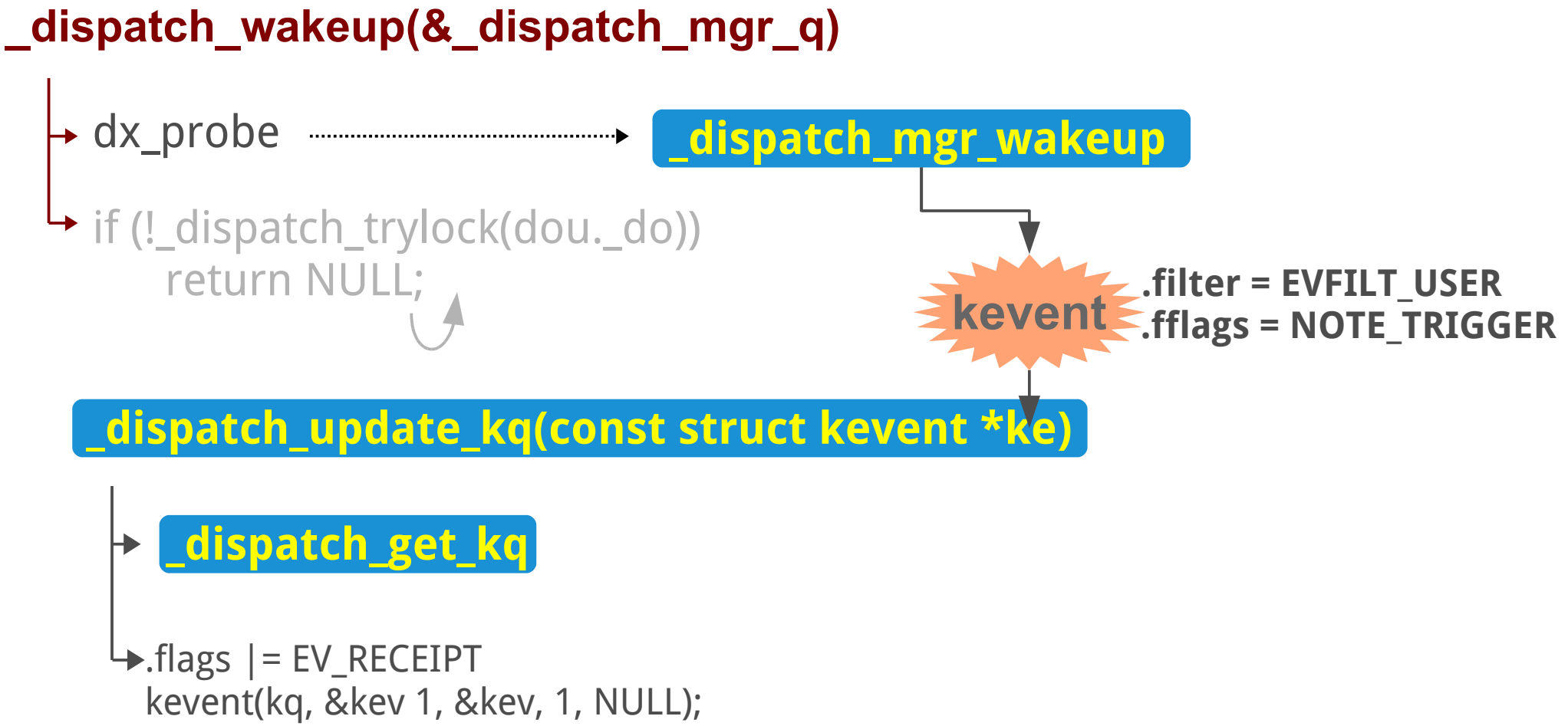
**_dispatch_wakeup(&_dispatch_mgr_q)**

- dx_probe ..... **_dispatch_mgr_wakeup**
- if (!_dispatch_trylock(dou._do))
  return NULL;

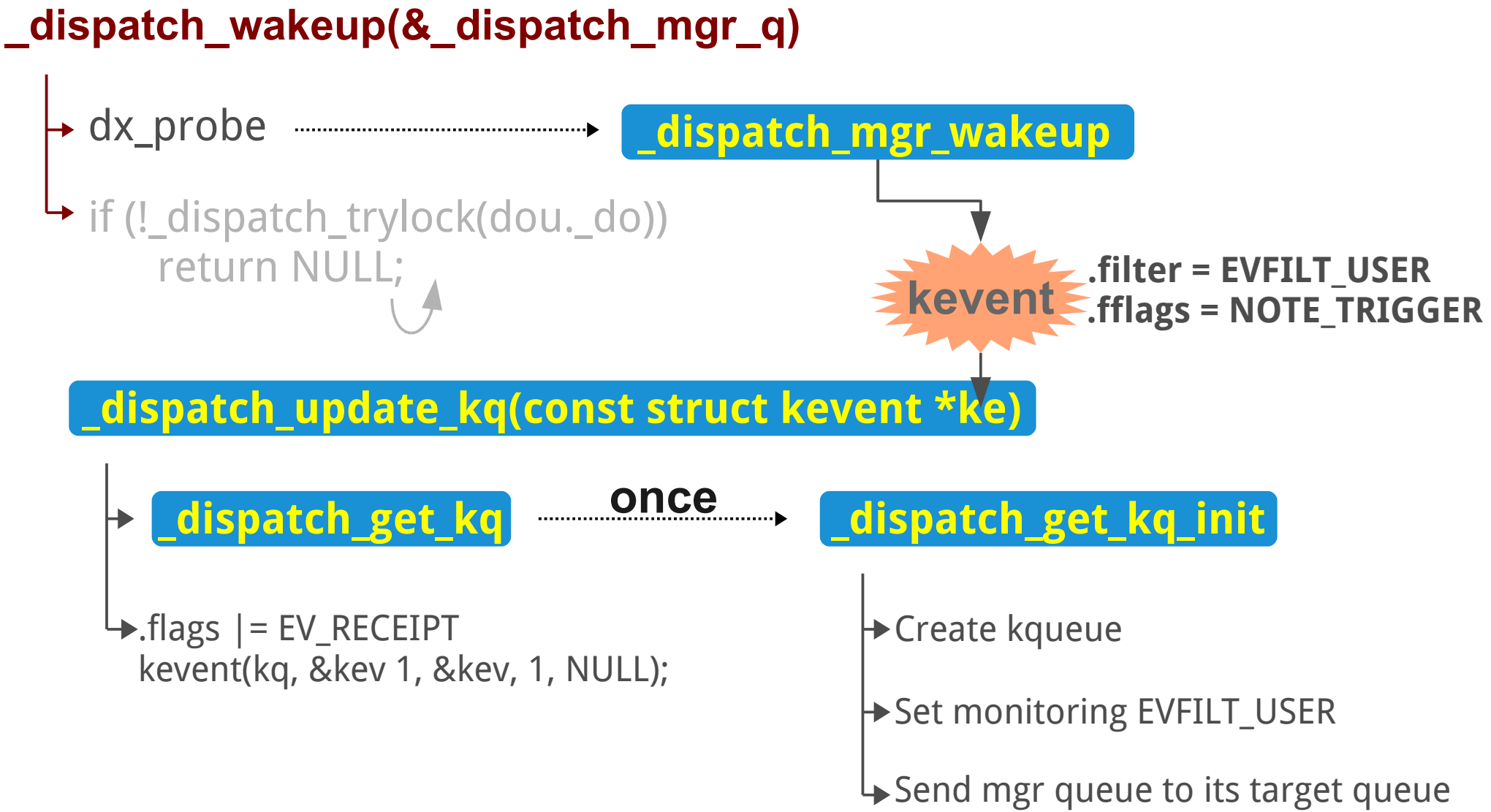
# Wake up mgr queue



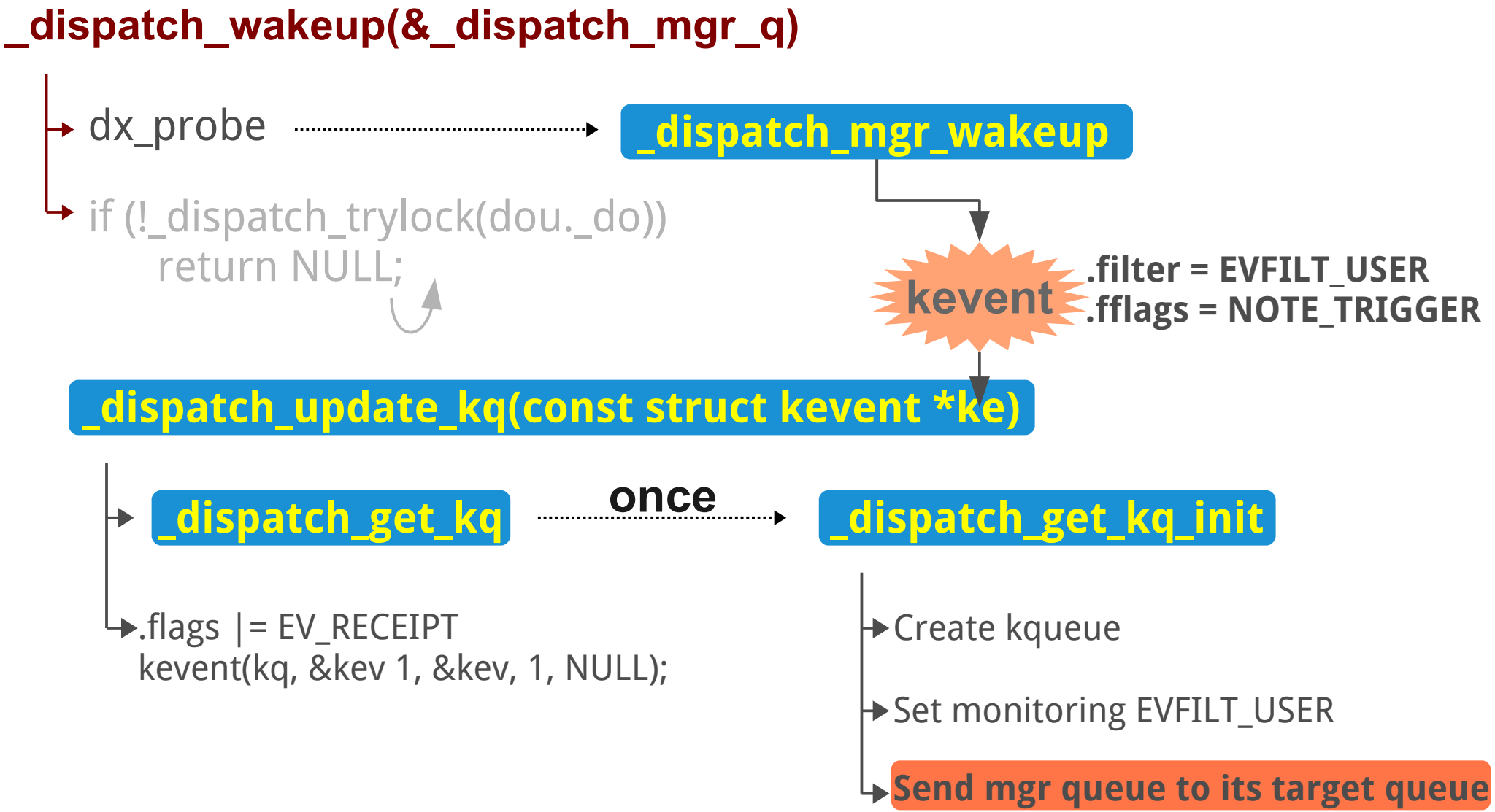
# Wake up mgr queue



# Wake up mgr queue



# Wake up mgr queue

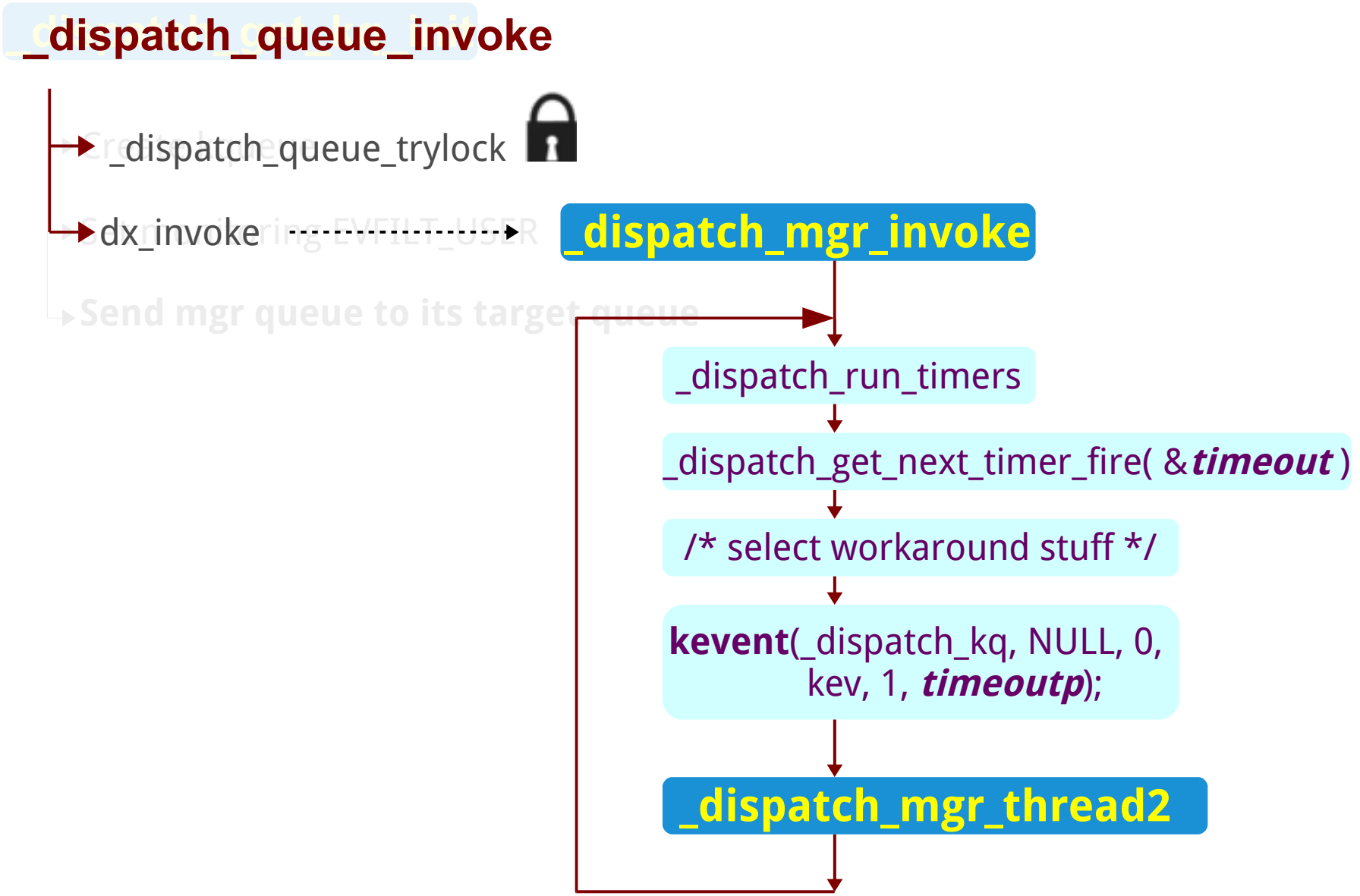


# Invoke mgr queue

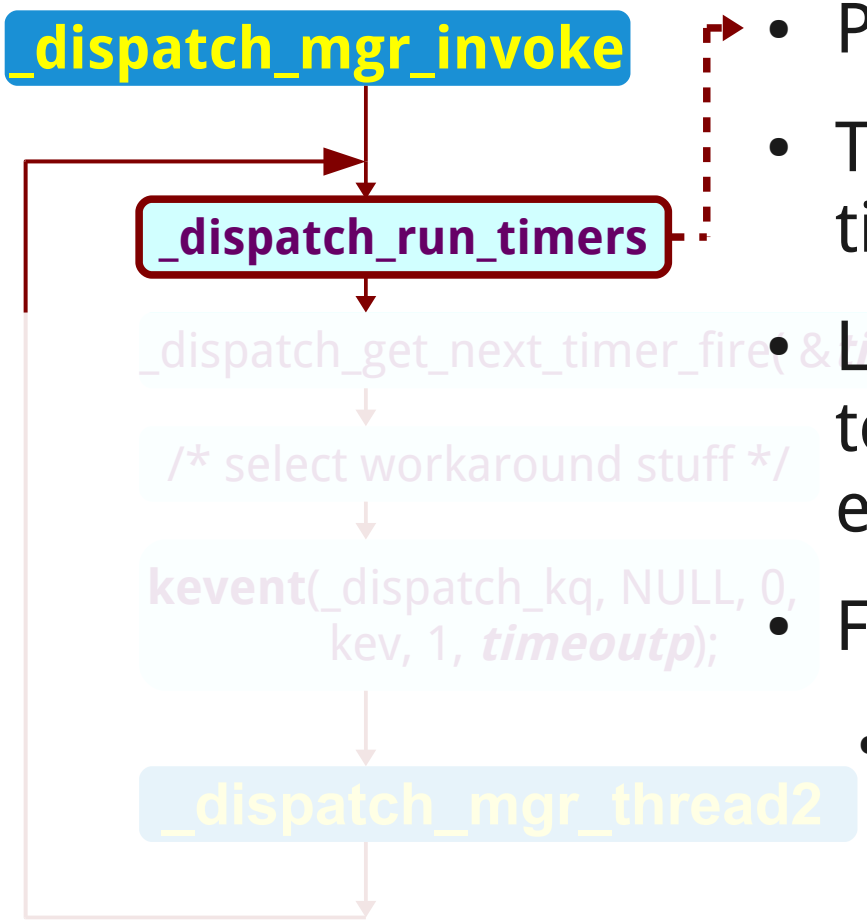
**_dispatch_get_kq_init**

- Create kqueue
- Set monitoring EVFILT_USER
- **Send mgr queue to its target queue** ⟹ **_dispatch_queue_invoke**

# Invoke mgr queue

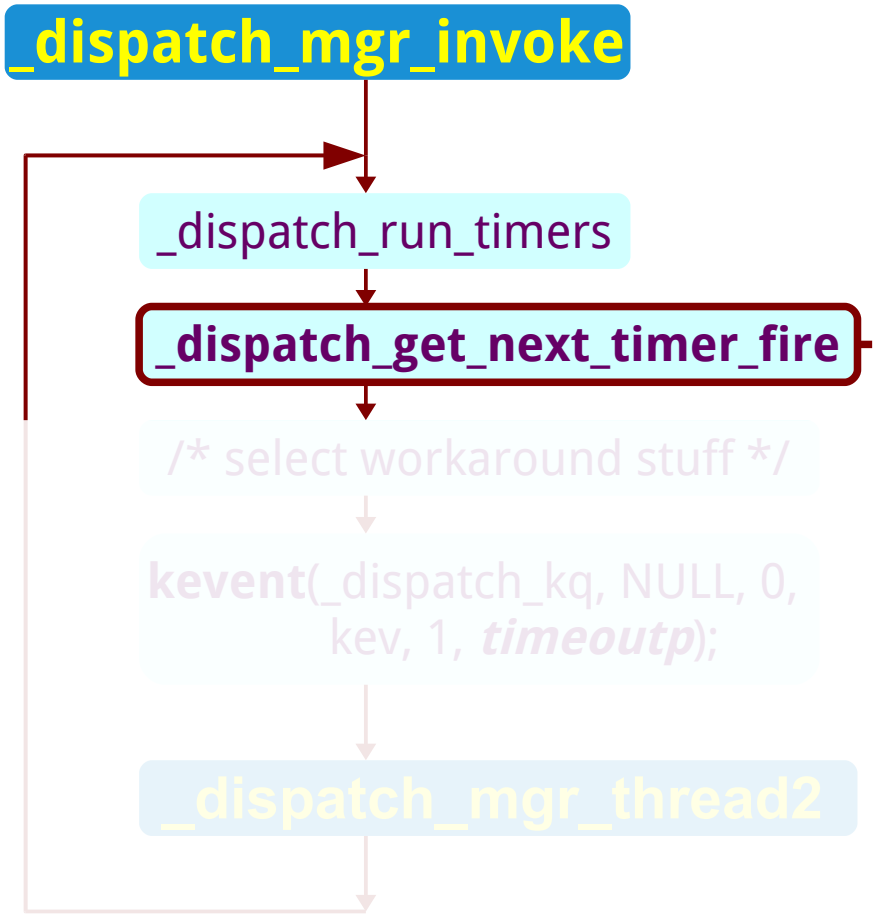


# Event loop – running timers




- Process timer sources
- Two types of timer: wall timer; abs timer
- Link timers of the same type together(by order of expiration time, early → late)
- For each timer list:
  - For each expired timer:
    - Early process
    - Update next expiration time
    - Adjust its position in list
    - Wake up it

# Event loop – Calc timeout




- Final timeout is 0 if exist an expired timer.
- Or calc by travel though two timer lists:
  1. Find the earlies expiration times of all vaild timers(not SUSPEND and has set expiration time) for each timer list
  2. Calc timeout from expiration time
  3. Convert abs timeout to wall timeout, find an smaller one

# Event loop – kevent


- A generic method of kernel event notification on BSD.
- Usage:
  - int kq = kqueue()
  - int ret = kevent(int kq,

    struct kevent *changelist,

    int nchanges,

    struct kevent *eventlist,

    int nevents,

    const struct timespec *timeout);

# struct kevent

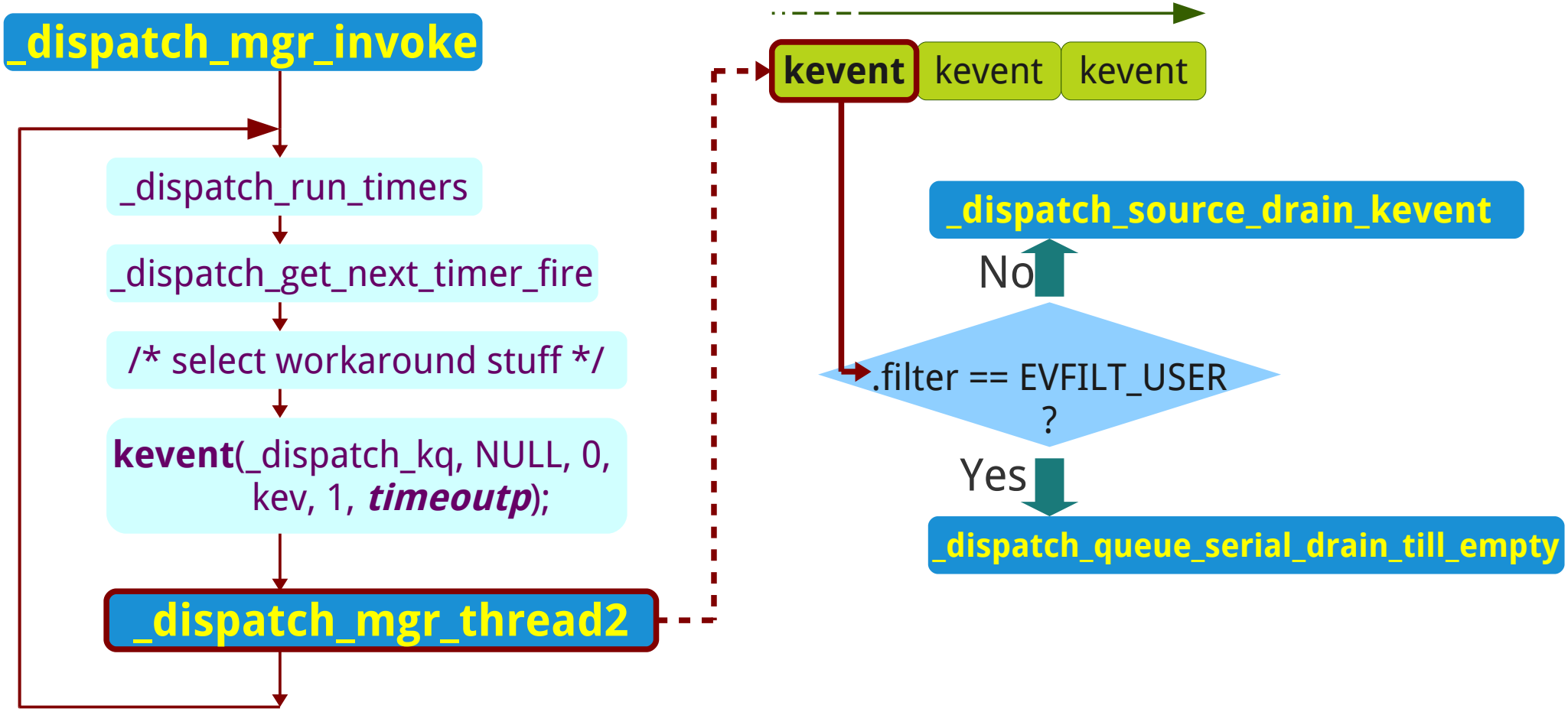
- struct kevent

| | | |
|---|---|---|
| uintptr_t | **ident;** | Value used to identify this event. The exact interpretation is determined by the attached filter, but often is a file descriptor. |
| int16_t | **filter**; | Identifies the kernel filter used to process this event. |
| uint16_t | **flags**; | Actions to perform on the event. |
| uint32_t | **fflags**; | Filter-specific flags. |
| intptr_t | **data**; | Filter-specific data value. |
| void | ***udata**; | Opaque user-defined value passed through the kernel unchanged. |

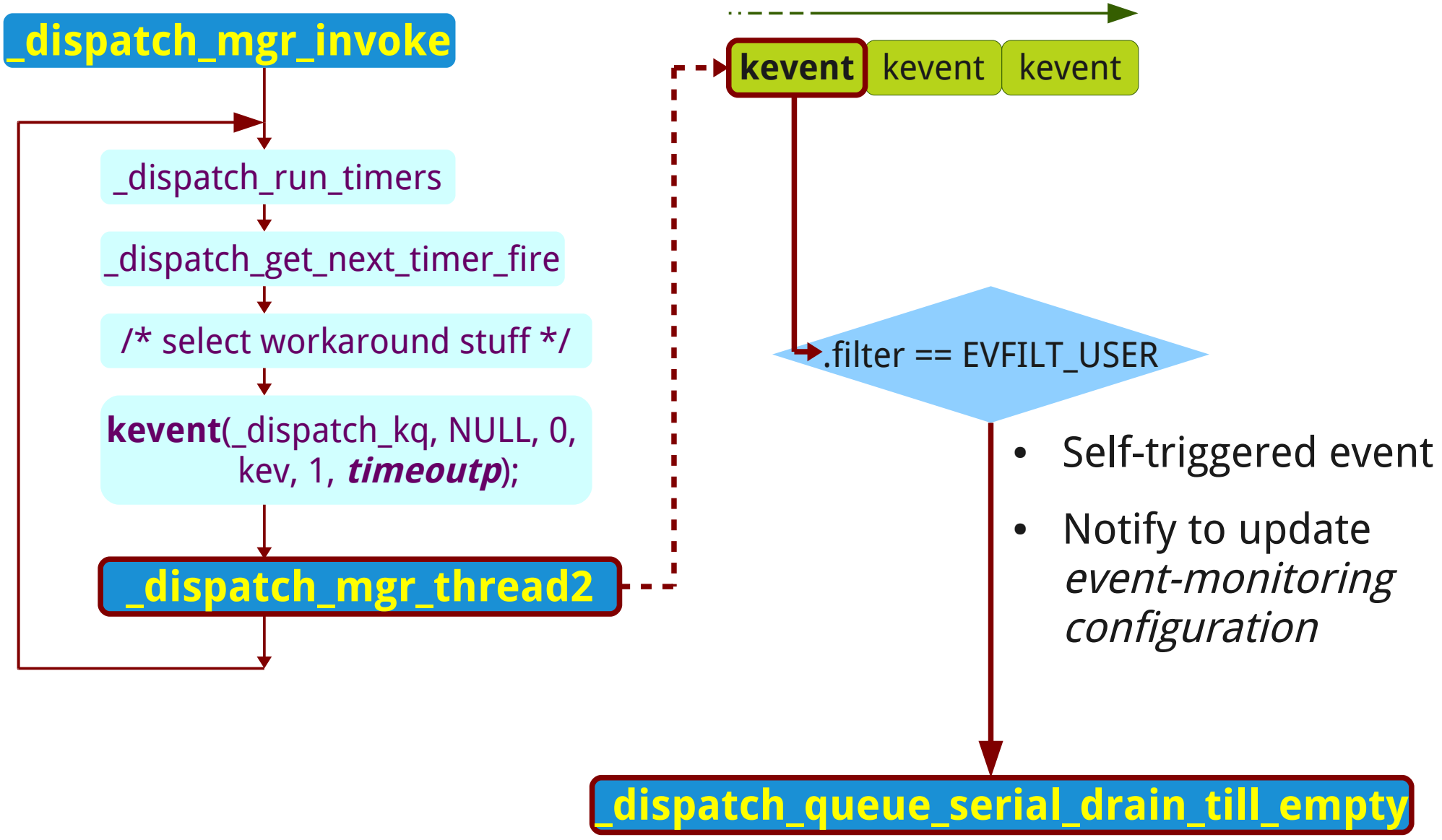
# filters

- EVFILT_READ — Data available for read on monitored fd
- EVFILT_WRITE — Buffer available for write on monitored fd
- EVFILT_VNODE — Notify operations on inode of monitored fd
- EVFILT_PROC
- EVFILT_SIGNAL
- EVFILT_TIMER
- ...

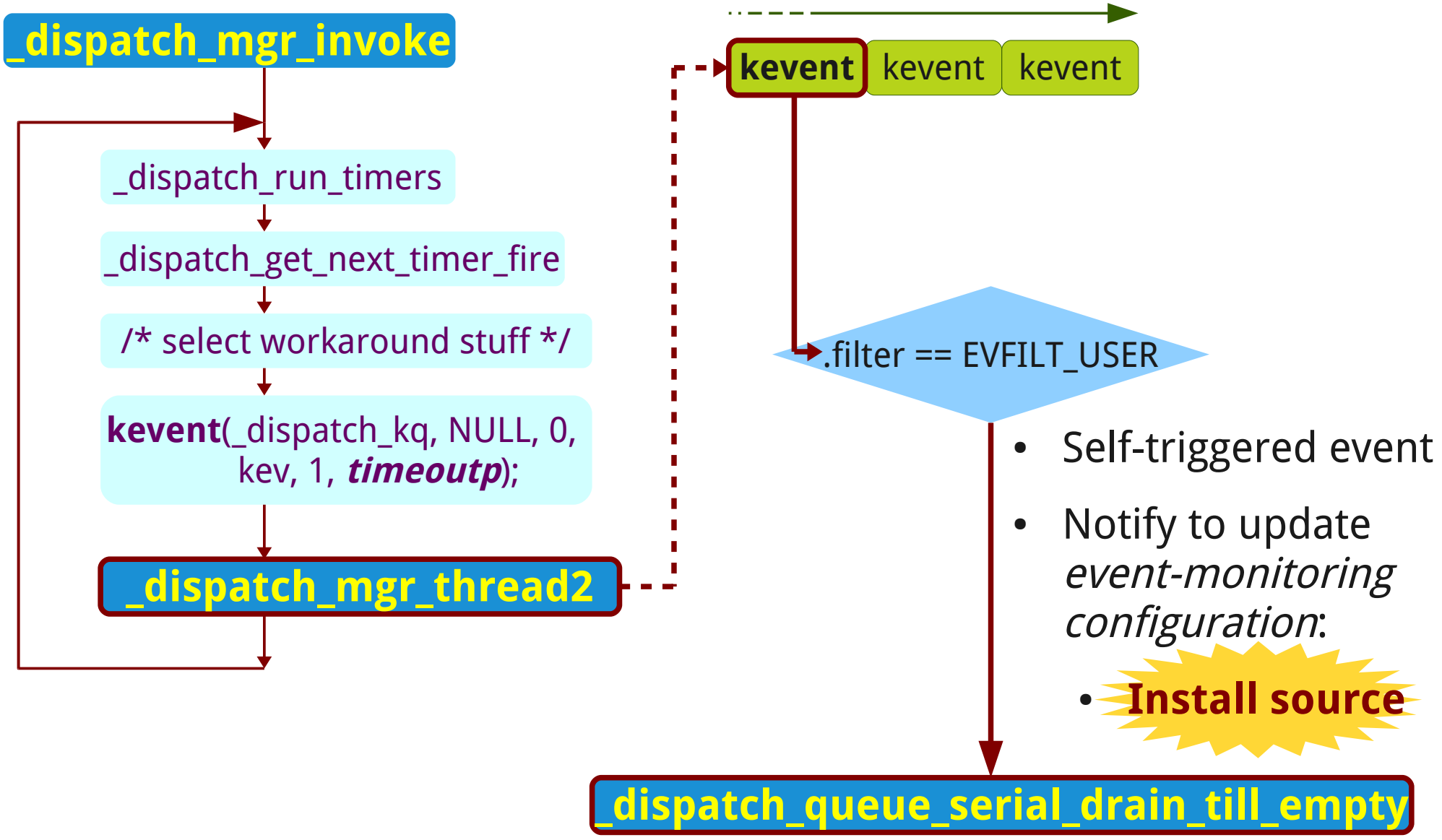
# Event loop – process event



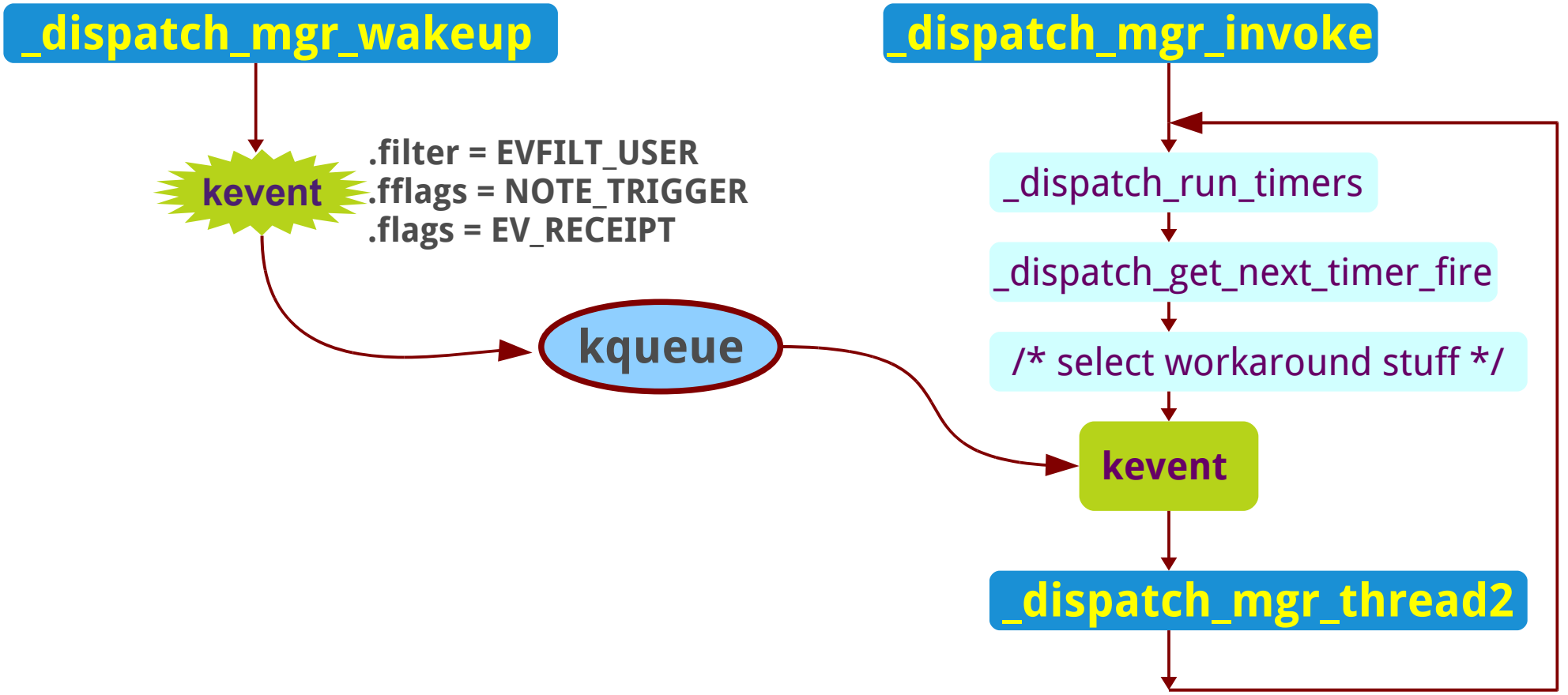
# Event loop – process event



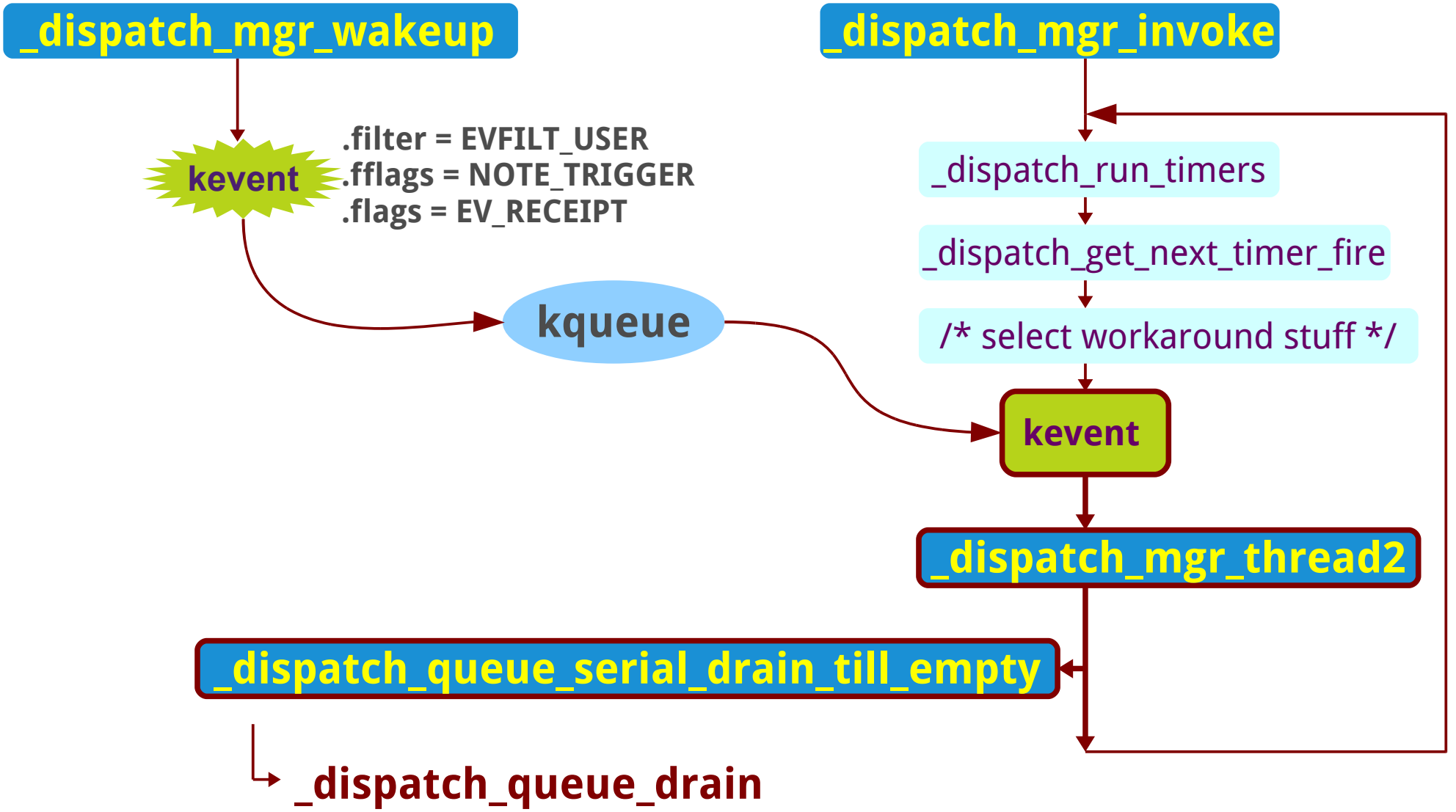
# Event loop – process event



# Run source(Installation stage2)



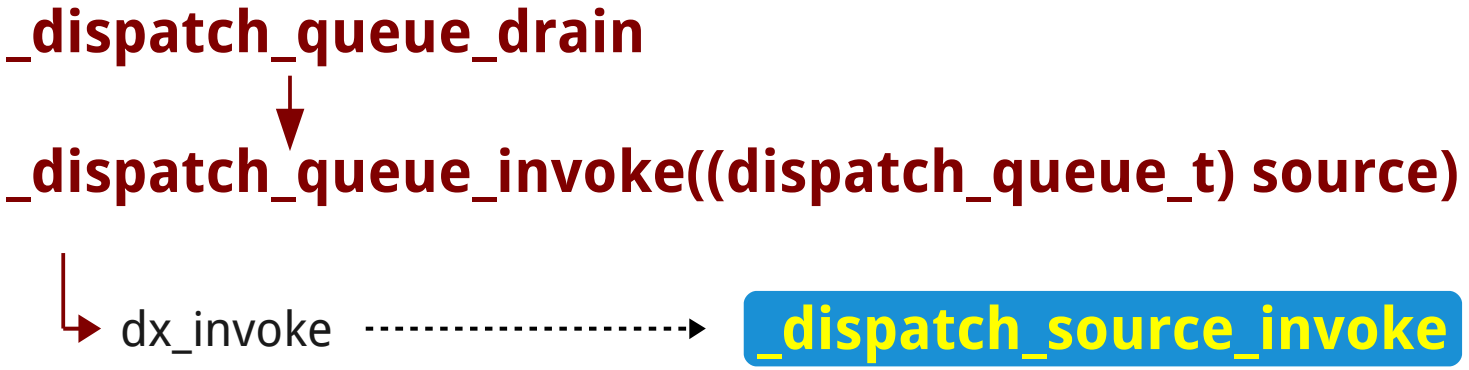
# Run source(Installation stage2)



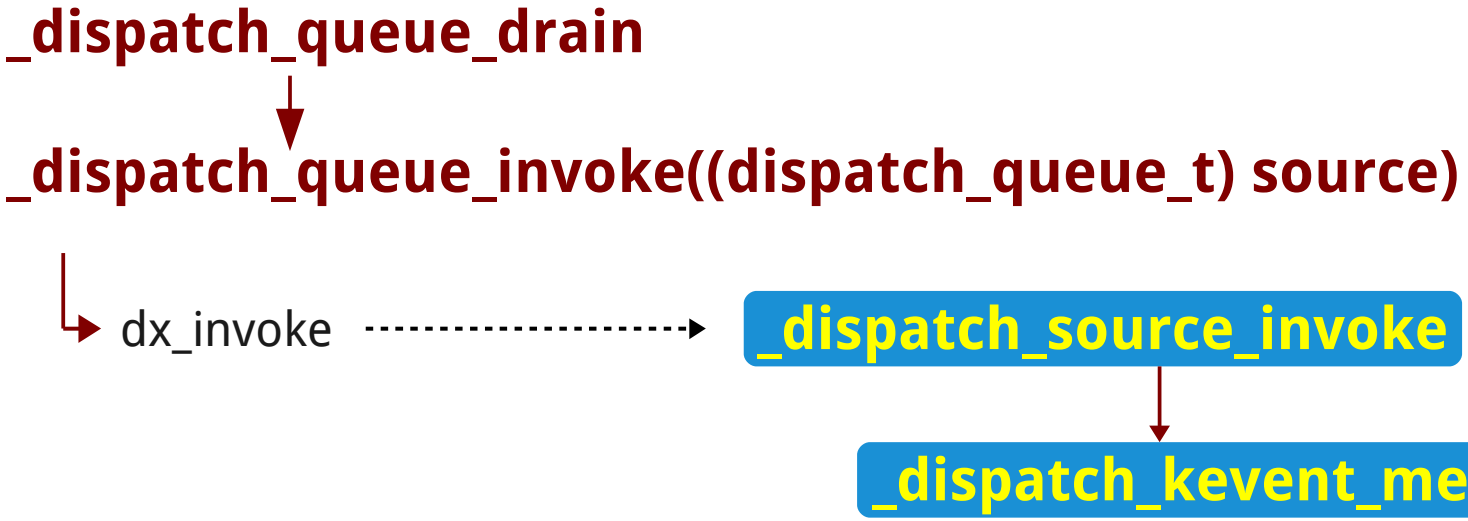
# Run source(Installation stage2)

**_dispatch_queue_drain**

# Run source(Installation stage2)

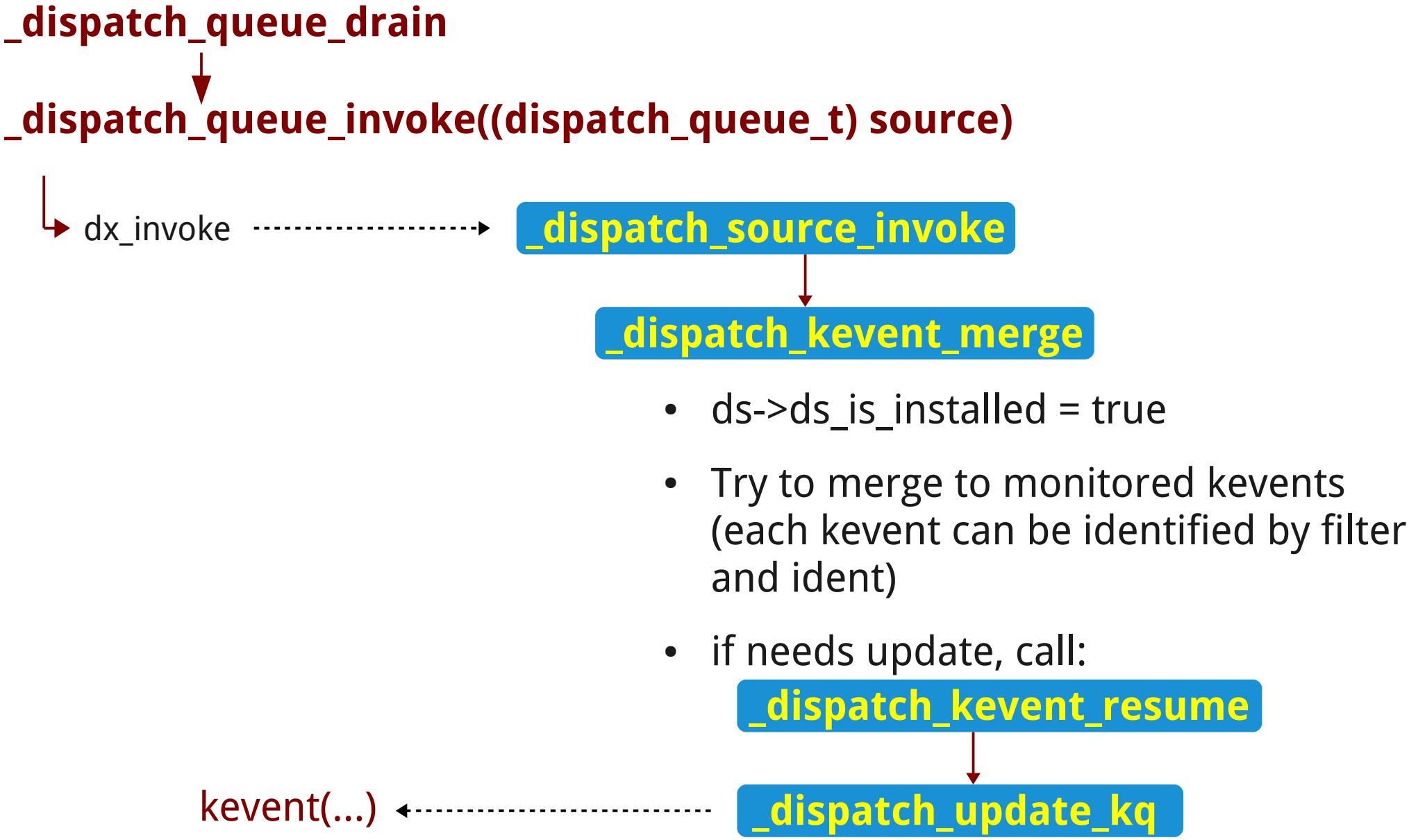


# Run source(Installation stage2)

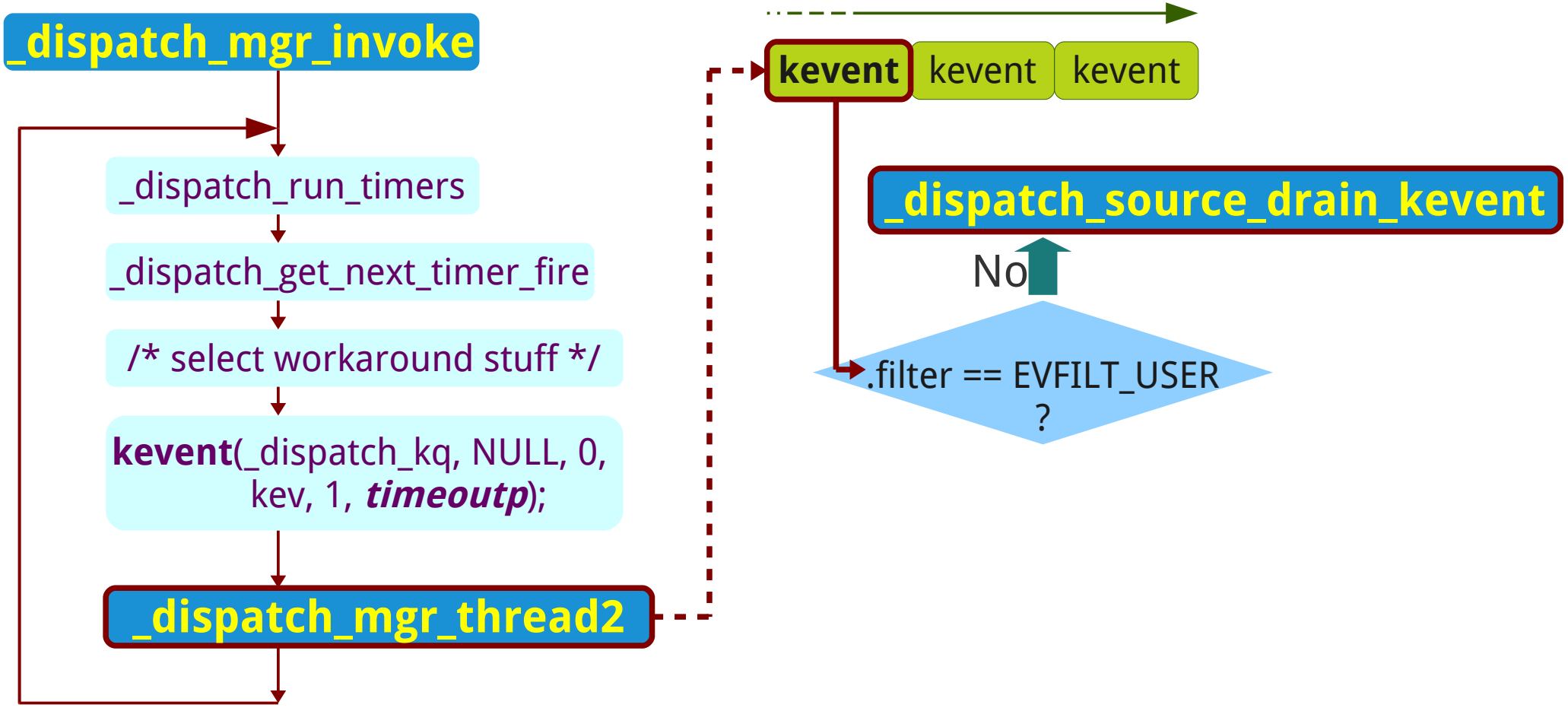



- ds->ds_is_installed = true
- Try to merge to monitored kevents (each kevent can be identified by filter and ident)
- if needs update, call:
  **_dispatch_kevent_resume**

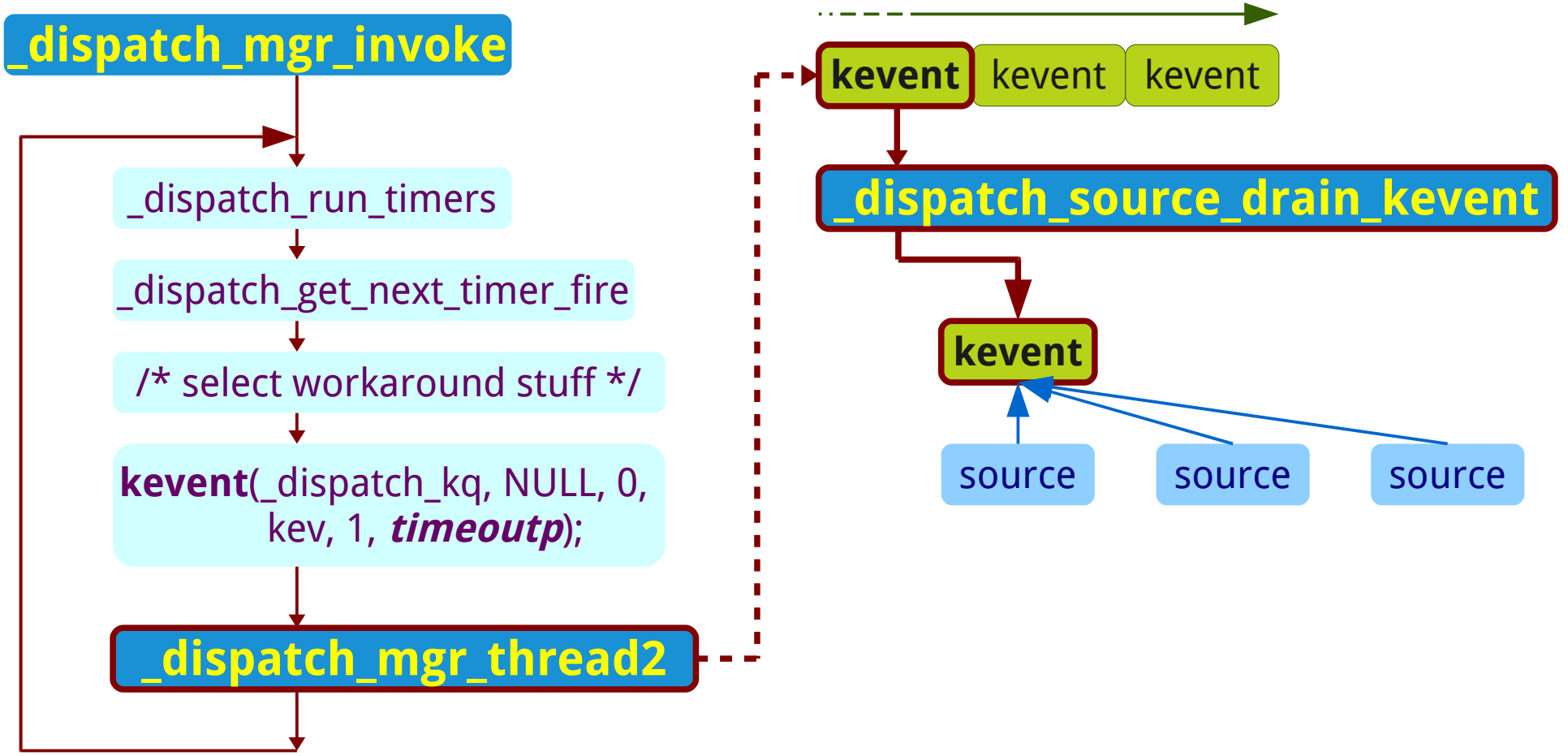
# Run source(Installation stage2)



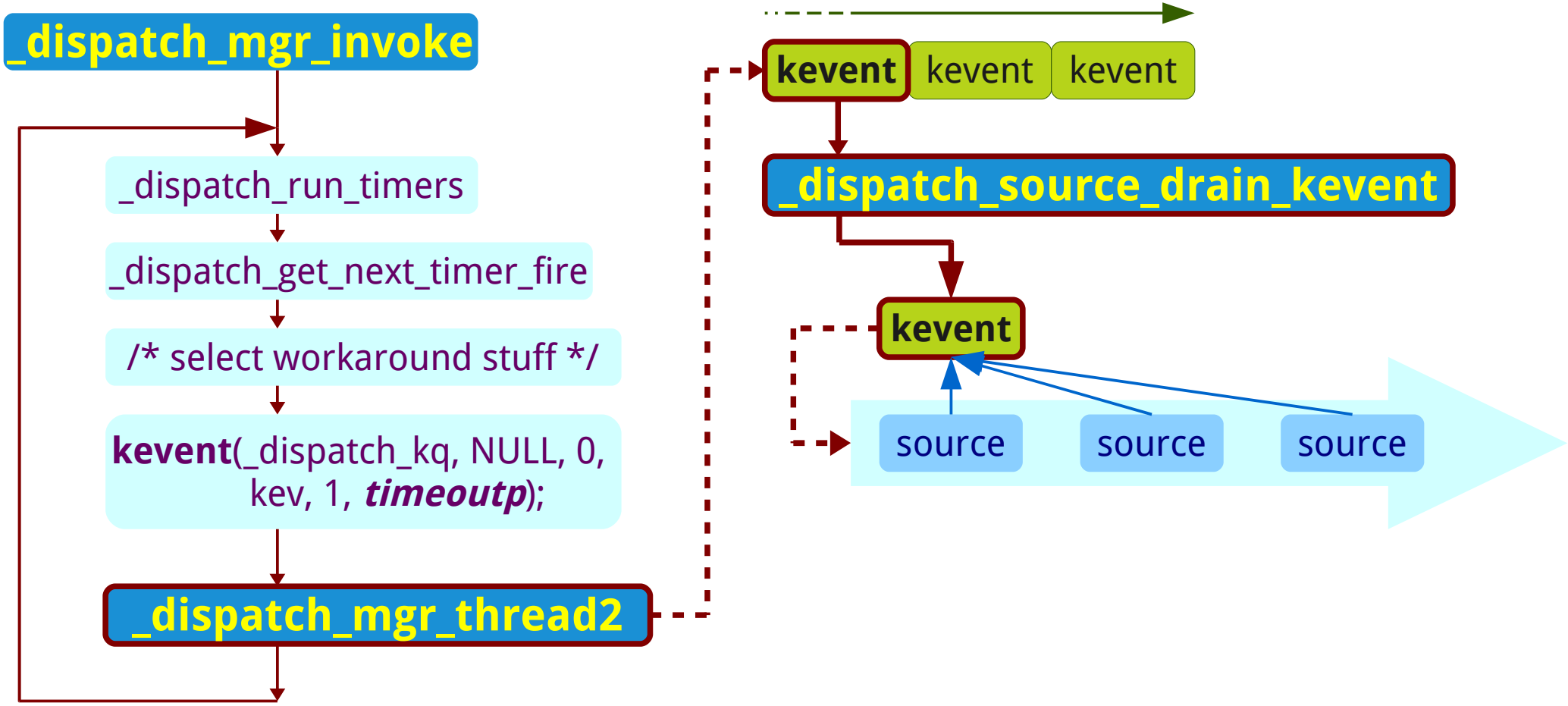
# Event loop – process event



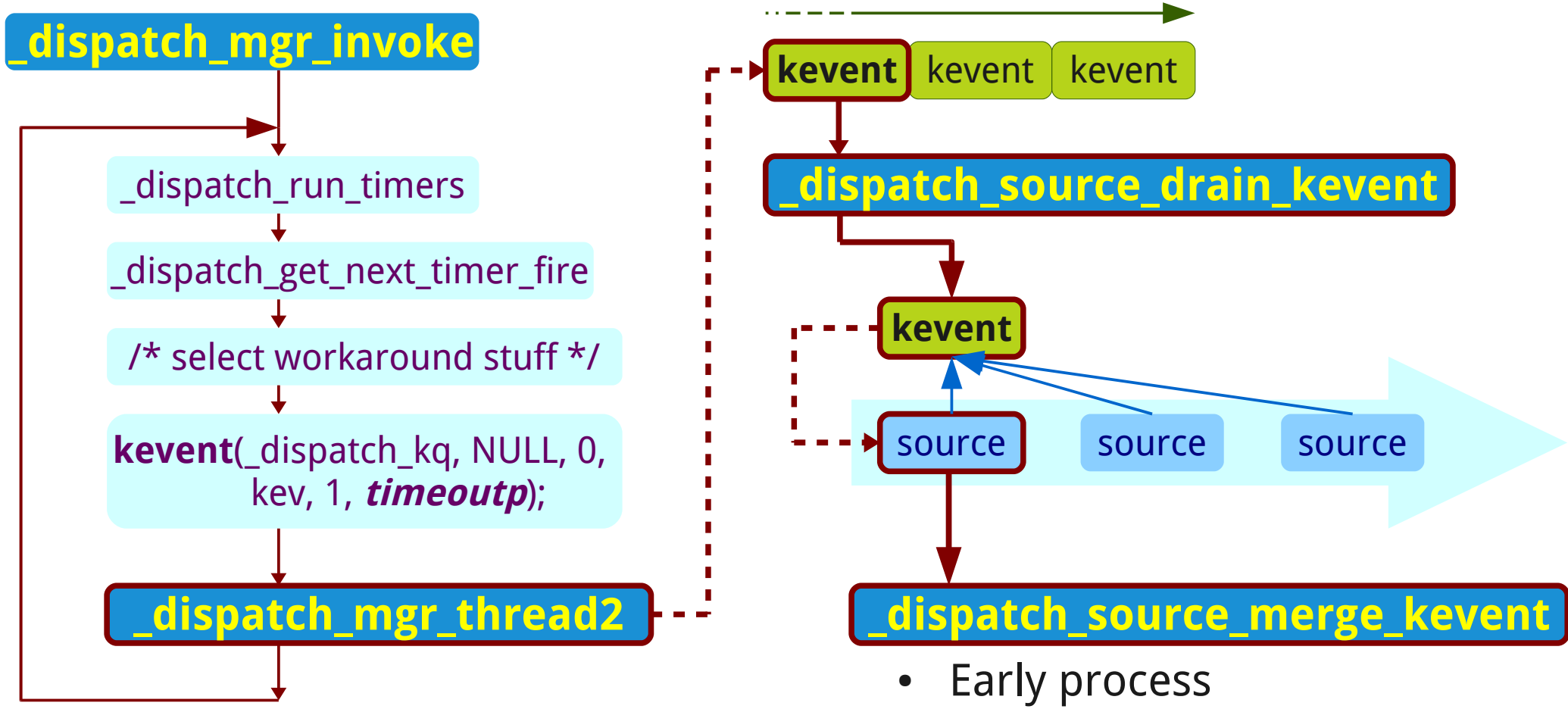
# Event loop – process event



# Event loop – process event



# Event loop – process event




- Early process
- Wake up source

# Process event(early)

- Classify events:
  - **level**: ds_pending_data = kev->data
    - DISPATCH_SOURCE_TYPE_READ
    - DISPATCH_SOURCE_TYPE_WRITE
  - **adder**: ds_pending_data += ke->data
    - DISPATCH_SOURCE_TYPE_SIGNAL
    - DISPATCH_SOURCE_TYPE_TIMER
  - **or**: ds_pending_data |= (kev->fflags & ds->ds_pending_data_mask)
    - DISPATCH_SOURCE_TYPE_VNODE

# Process event

- Wake up source → send to its target queue
- **_dispatch_queue_drain**

  **_dispatch_queue_invoke((dispatch_queue_t) source)**

  dx_invoke ----------> **_dispatch_source_invoke**

  **_dispatch_source_latch_and_call**

  - ds_handler_func(ds_handler_ctxt);

END