SSL INTERCEPT PRACTICE

Oknet (Chao Xu)
xuchao@skyguard.com.cn
oknet@apache.org

SkyGuard

- Since Jan 2015
- Products / Solutions
  - Inside Threat Protection
  - Data Loss Prevention
  - UEBA
  - CASB
  - Secure Email Gateway
  - Secure Web Gateway (**based on ATS**)

天空卫士
SkyGuard

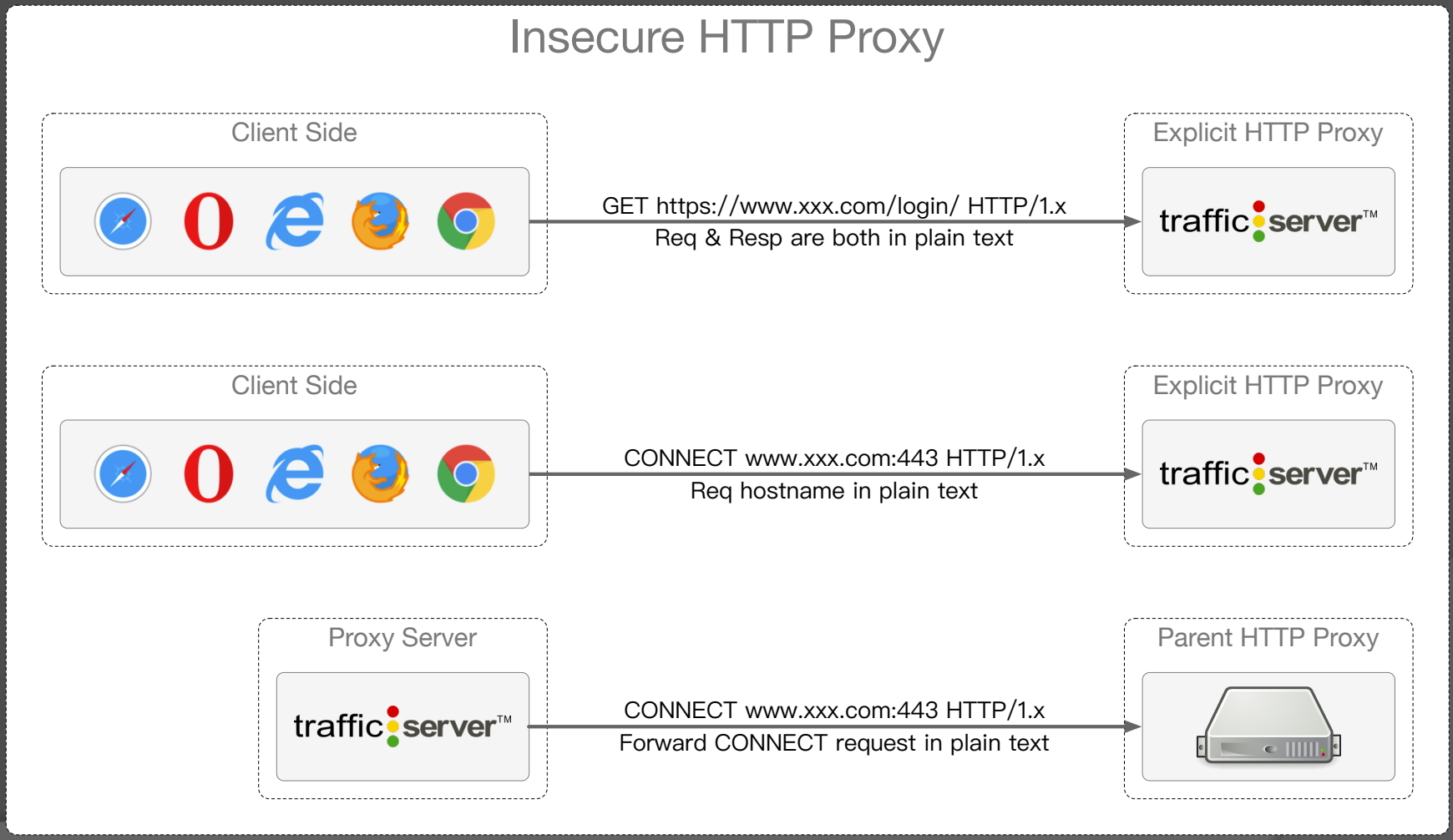
About me

- Jan 2015
  - Touch the ATS code
  - Review code line by line and write code reading note on Github
  - https://github.com/oknet/atsinternals
- Sep 2016
  - ATS committer and PMC member
- Mainly focus on the stability and quality of the code, then performance.

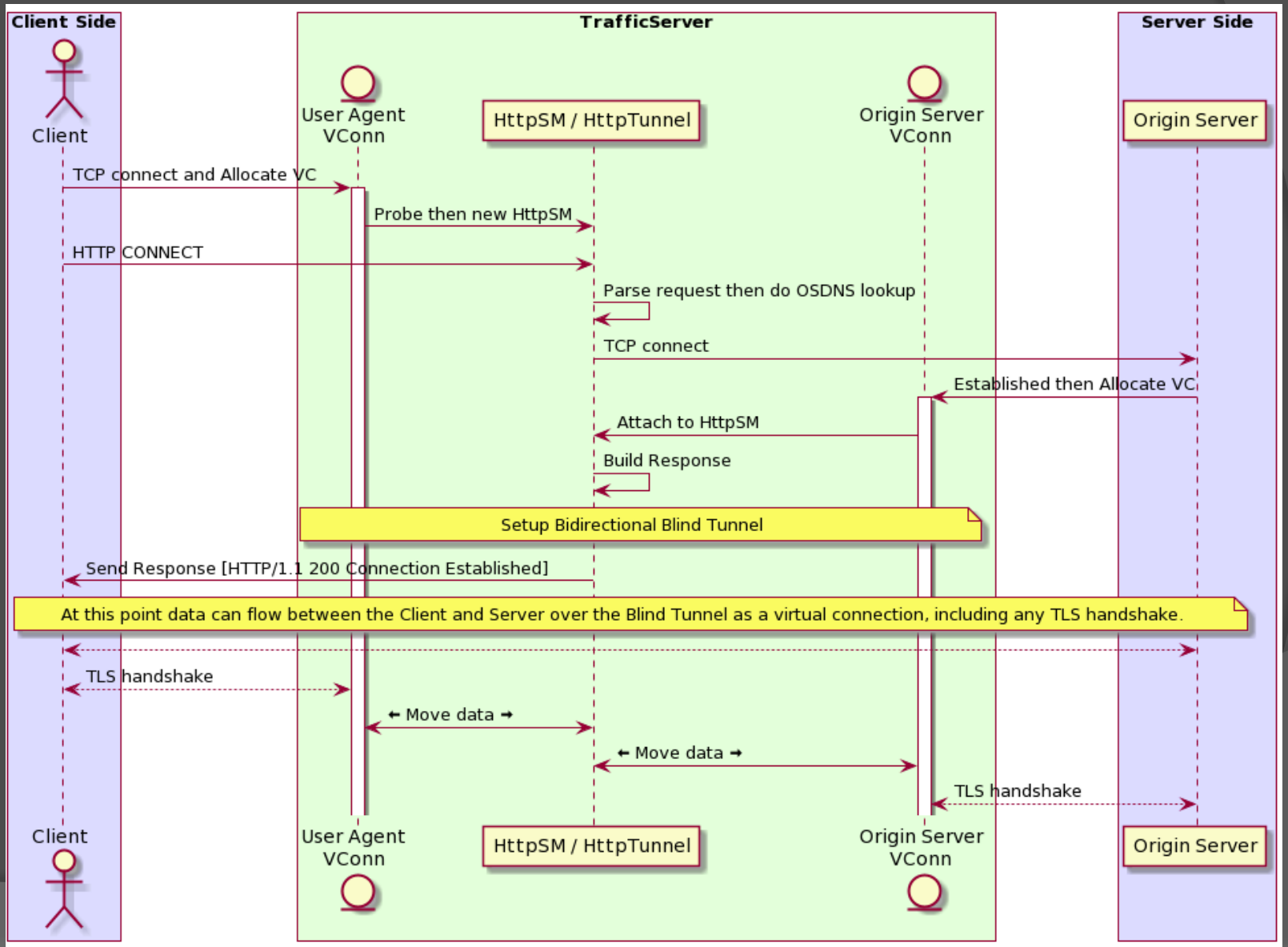
Start from

The Explicit Proxy

Insecure HTTP Proxy



Deal with CONNECT method



Deal with CONNECT method

```
@startuml
  box "Client Side" #DDDDFF
  actor Client
  end box

  box "TrafficServer" #DDFFDD
  entity "User Agent\nVConn" as lvc
  participant "HttpSM / HttpTunnel" as httpsm
  entity "Origin Server\nVConn" as rvc
  end box
  box "Server Side" #DDDDFF
  participant "Origin Server" as Server
  end box

  Client -> lvc : TCP connect and Allocate VC
  activate lvc
  lvc -> httpsm : Probe then new HttpSM
  Client -> httpsm : HTTP CONNECT
  httpsm -> httpsm : Parse request then do OSDNS lookup
  httpsm -> Server : TCP connect
  Server -> rvc : Established then Allocate VC
  activate rvc
  rvc -> httpsm : Attach to HttpSM
  httpsm -> httpsm : Build Response
  note over lvc,rvc : Setup Bidirectional Blind Tunnel
  httpsm -> Client : Send Response [HTTP/1.1 200 Connection Established]


  note over Client, Server : At this point data can flow between the Client and Server
  over the Blind Tunnel as a virtual connection, including any TLS handshake.
  Client <--> Server
  Client <--> lvc : TLS handshake
  lvc <-> httpsm : <&arrow-thick-left> Move data <&arrow-thick-right>
  httpsm <-> rvc : <&arrow-thick-left> Move data <&arrow-thick-right>
  rvc <--> Server : TLS handshake
@enduml
```

Intercept bidirectional tunnel

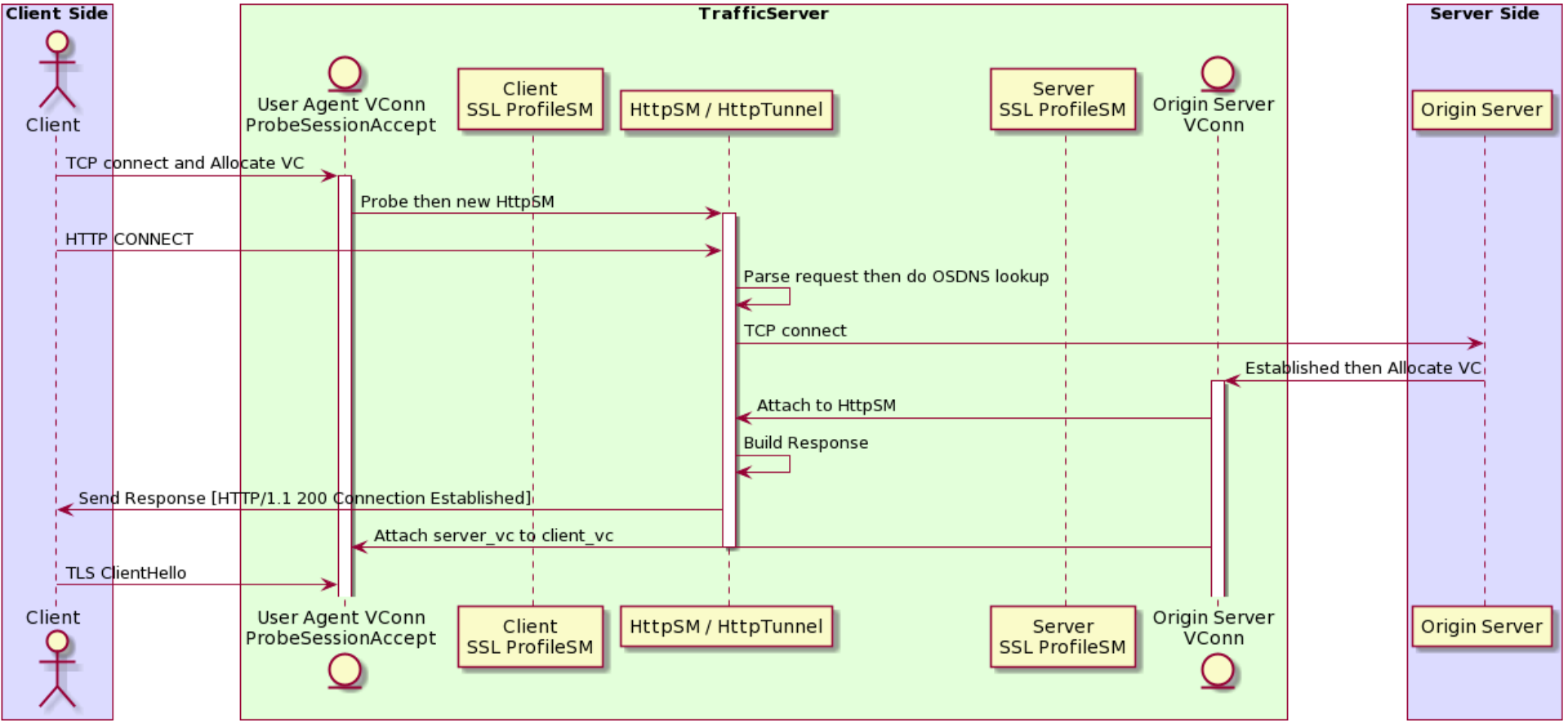
Why

- Did the employee
  - download malware ?
  - receive phishing email ?
  - visit a malicious website ?
  - leak important information ?
- For the risk
  - Detect and prevent early.

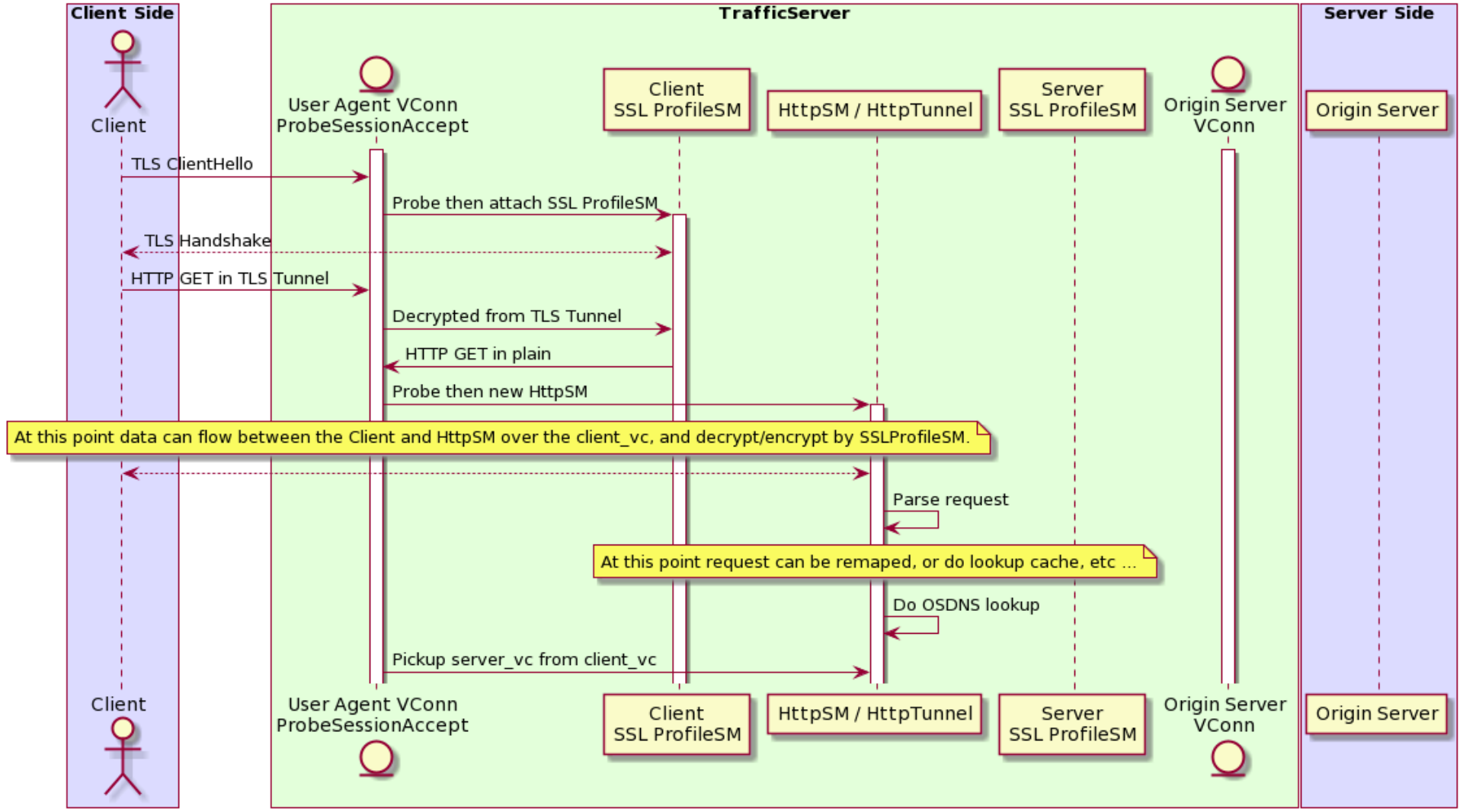
How

- Extend ProtocolProbeSessionAccept
  - Learn from IDS
  - Detect and identify certain protocols by signatures. Includes:
    - SSLv2, SSLv3, TLS1, …
    - Socksv4, Socksv5
  - Detect and identify certain protocols by tcp ports. Includes:
    - FTP(21), FTPS(990)
    - SMTP(25, 587), POP3(110), IMAP(143), …
    - SMTPS(465), POP3S(995), IMAPS(993), …
  - Combine the two methods to get more accurate result
- Introduce ProfileSM
  - Pull low level I/O operations from UnixNetVC
  - Pull SSL handshake and encrypt/decrypt from SSLNetVC
  - Create tcpProfileSM and sslProfileSM as NetVC's helper to perform these operations.

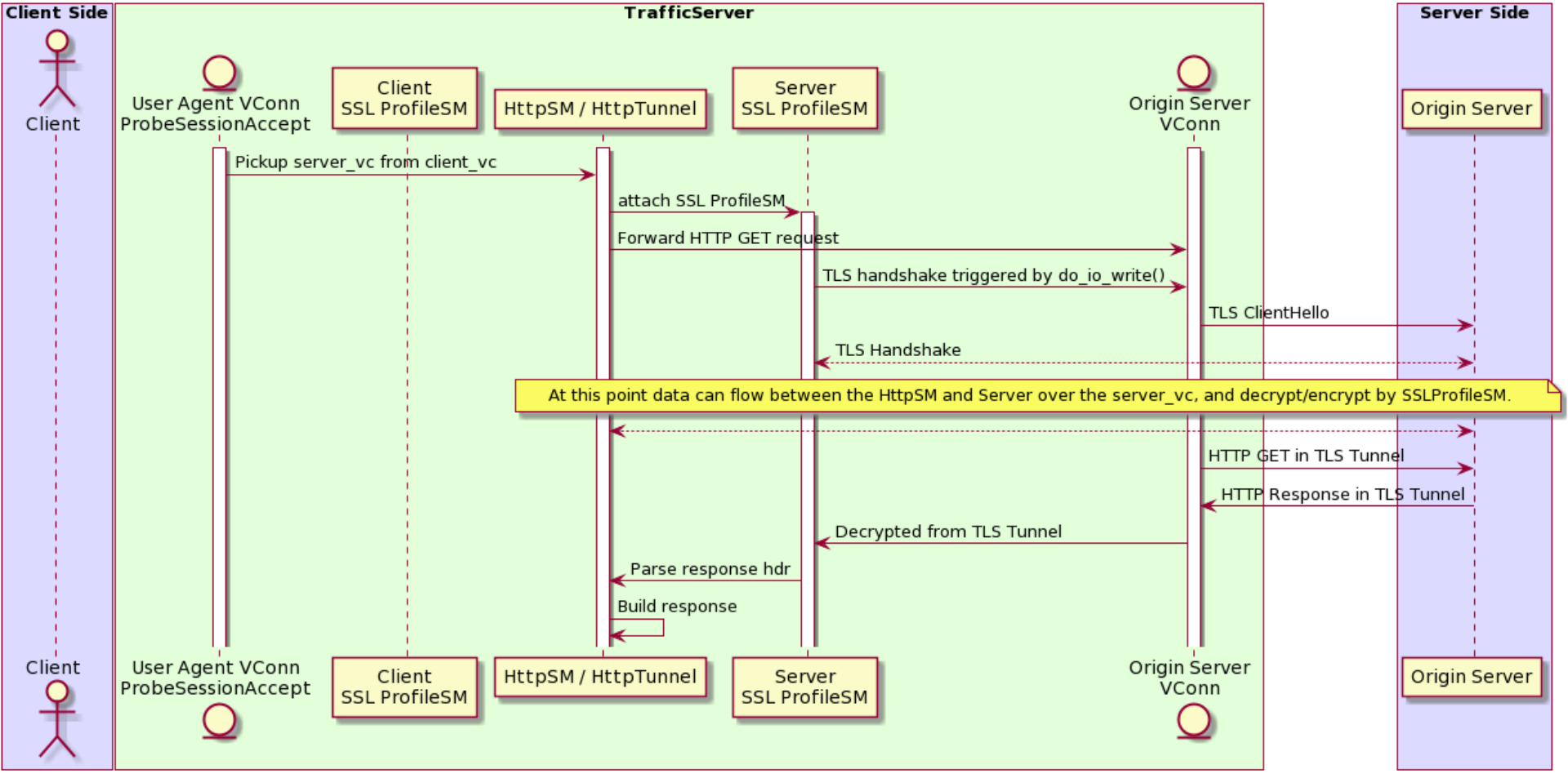
Handle SSL from Blind Tunnel 1



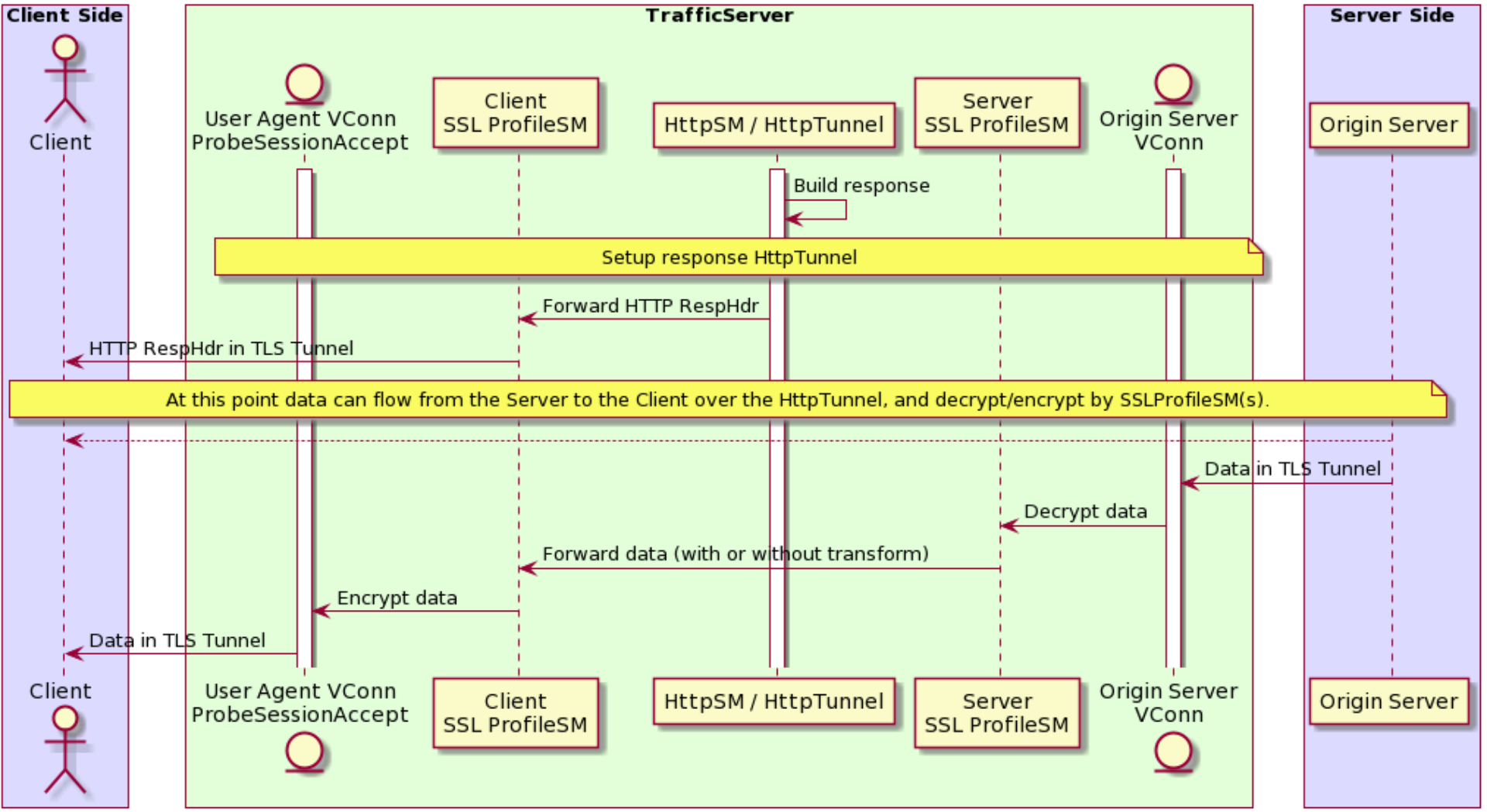
Handle SSL from Blind Tunnel 2



Handle SSL from Blind Tunnel 3



Handle SSL from Blind Tunnel 4



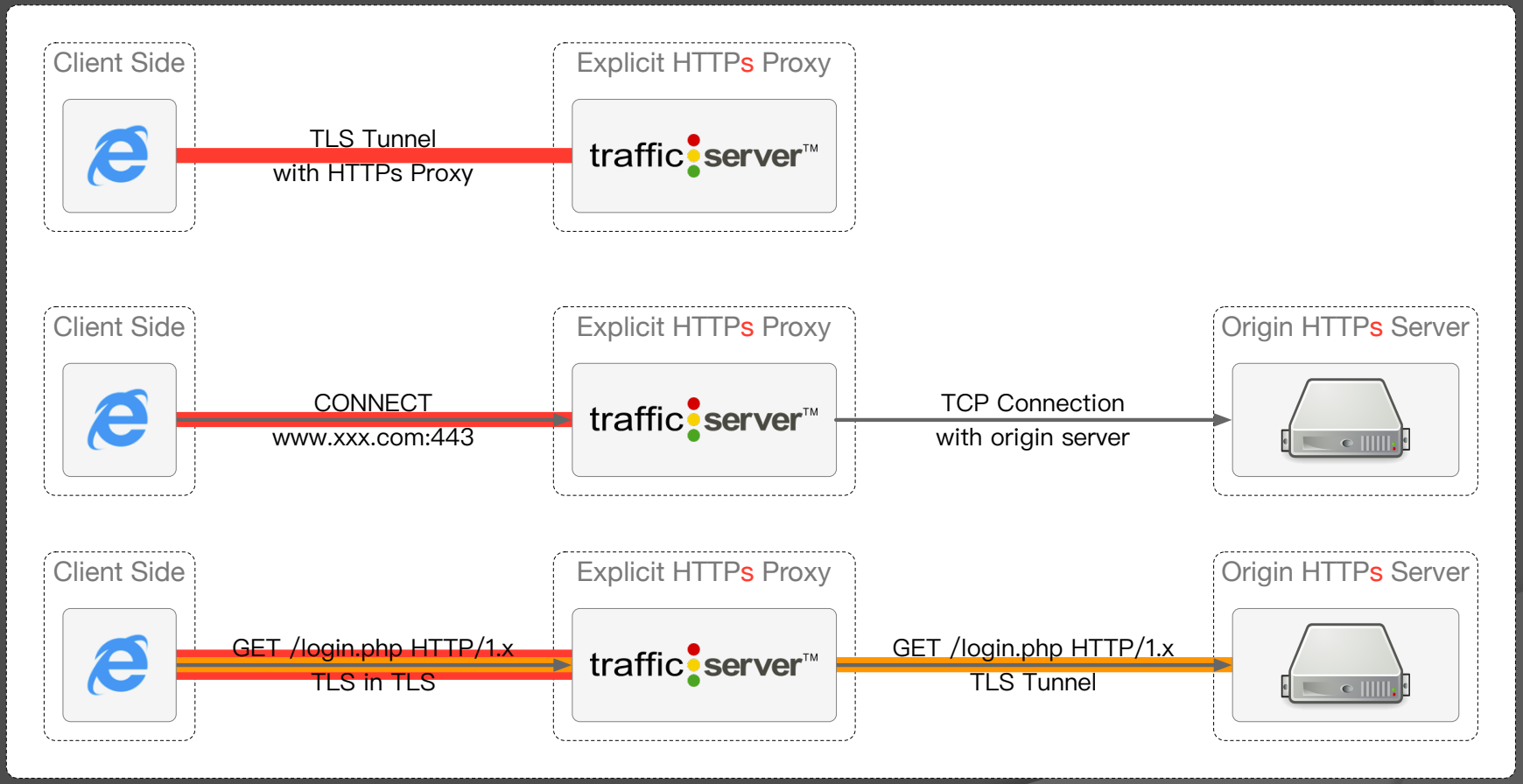
Handle SSL from Blind Tunnel with SslProfileSM

```
@startuml
  box "Client Side" #DDDDFF
  actor Client
  end box

  box "TrafficServer" #DDFFDD
  entity "User Agent VConn\nProbeSessionAccept" as lvc
  participant "Client\nSSL ProfileSM" as lvcssl
  participant "HttpSM / HttpTunnel" as httpsm
  participant "Server\nSSL ProfileSM" as rvcssl
  entity "Origin Server\nVConn" as rvc
  end box
  box "Server Side" #DDDDFF
  participant "Origin Server" as Server
  end box

  Client -> lvc : TCP connect and Allocate VC
  activate lvc
  lvc -> httpsm : Probe then new HttpSM
  activate httpsm
  Client -> httpsm : HTTP CONNECT
  httpsm -> httpsm : Parse request then do OSDNS lookup
  httpsm -> Server : TCP connect
  Server -> rvc : Established then Allocate VC
  activate rvc
  rvc -> httpsm : Attach to HttpSM
  httpsm -> httpsm : Build Response
  httpsm -> Client : Send Response [HTTP/1.1 200 Connection Established]
  rvc -> lvc : Attach server_vc to client_vc
  deactivate httpsm
  Client -> lvc : TLS ClientHello
  lvc -> lvcssl : Probe then attach SSL ProfileSM
  activate lvcssl
  Client <-->lvcssl : TLS Handshake
  Client -> lvc : HTTP GET in TLS Tunnel
  lvc -> lvcssl : Decrypted from TLS Tunnel
```

```
  lvcssl -> lvc : HTTP GET in plain
  lvc -> httpsm : Probe then new HttpSM
  activate httpsm
  note over Client, httpsm : At this point data can flow between the Client and HttpSM
  over the client_vc, and decrypt/encrypt by SSLProfileSM.
  Client <--> httpsm
  httpsm -> httpsm : Parse request
  note over httpsm : At this point request can be remaped, or do lookup cache, etc ...
  httpsm -> httpsm : Do OSDNS lookup
  lvc -> httpsm : Pickup server_vc from client_vc
  httpsm -> rvcssl : attach SSL ProfileSM
  activate rvcssl
  httpsm -> rvc : Forward HTTP GET request
  rvcssl -> rvc : TLS handshake triggered by do_io_write()
  rvc -> Server : TLS ClientHello
  rvcssl <--> Server : TLS Handshake
  note over httpsm, Server : At this point data can flow between the HttpSM and Server
  over the server_vc, and decrypt/encrypt by SSLProfileSM.
  httpsm <--> Server
  rvc -> Server : HTTP GET in TLS Tunnel
  Server -> rvc : HTTP Response in TLS Tunnel
  rvc -> rvcssl : Decrypted from TLS Tunnel
  rvcssl -> httpsm : Parse response hdr
  httpsm -> httpsm : Build response
  note over lvc, rvc : Setup response HttpTunnel
  httpsm -> lvcssl : Forward HTTP RespHdr
  lvcssl -> Client : HTTP RespHdr in TLS Tunnel
  note over Client, Server : At this point data can flow from the Server to the Client over
  the HttpTunnel, and decrypt/encrypt by SSLProfileSM(s).
  Client <-- Server
  rvc <- Server : Data in TLS Tunnel
  rvcssl <- rvc : Decrypt data
  lvcssl <- rvcssl : Forward data (with or without transform)
  lvc <- lvcssl : Encrypt data
  Client <- lvc : Data in TLS Tunnel
@enduml
```

Secure HTTP Proxy

TLS Tunnel in TLS Tunnel



SSLinSSLNetVConnection ?

Use ProfileSM to

Intercept SSL in SSL Tunnel

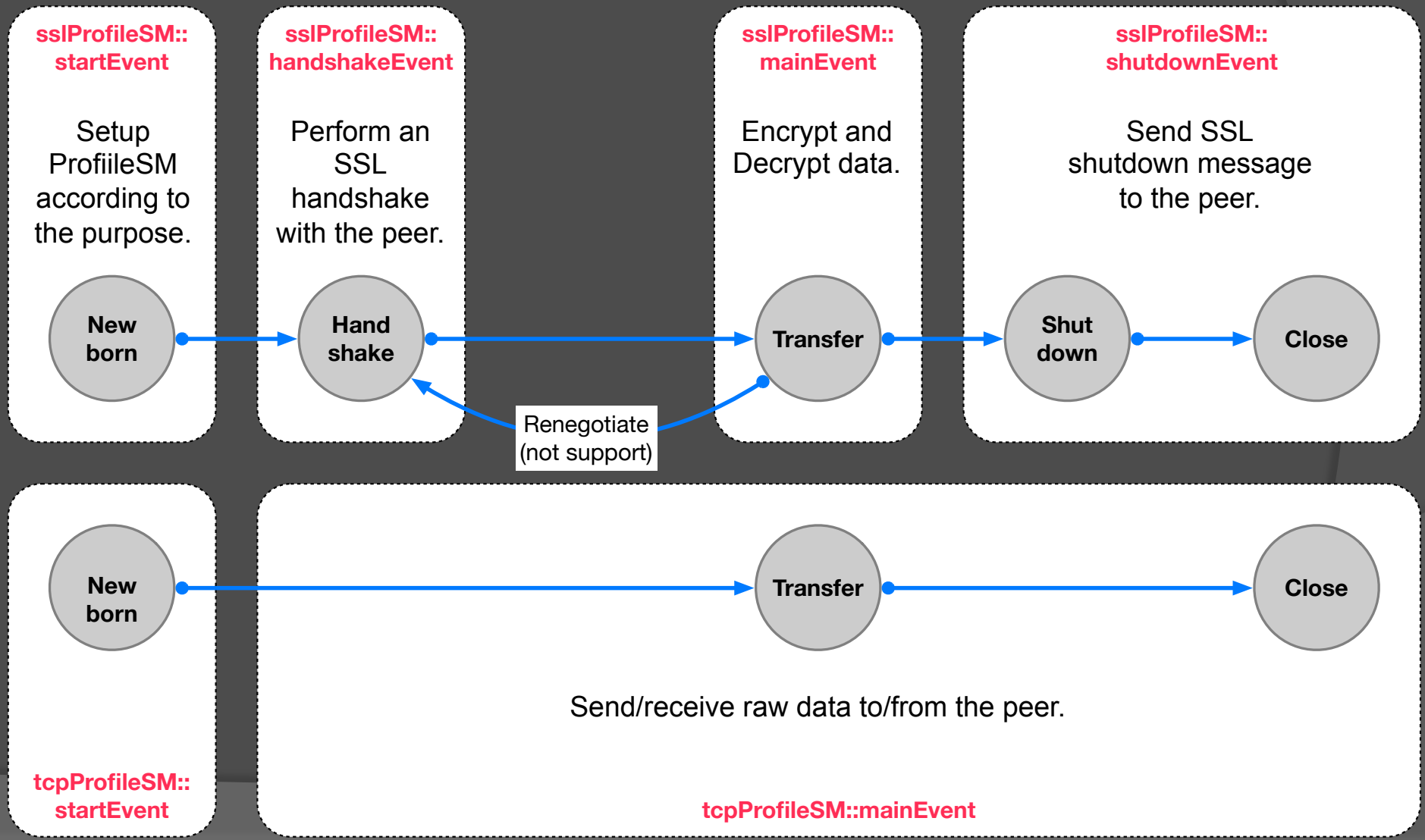
How

- SSL Read
  - Decrypt the encrypted content
- SSL Write
  - Encrypt the plaintext content
- sslProfileSM
  - Handle SSL Handshake
  - Data transform between NetVC and HttpSM
- It is just a content transform operation, but there are some differences:
  - Bidirectional (Read and Write).
  - Stateful (Handshake and Transfer).

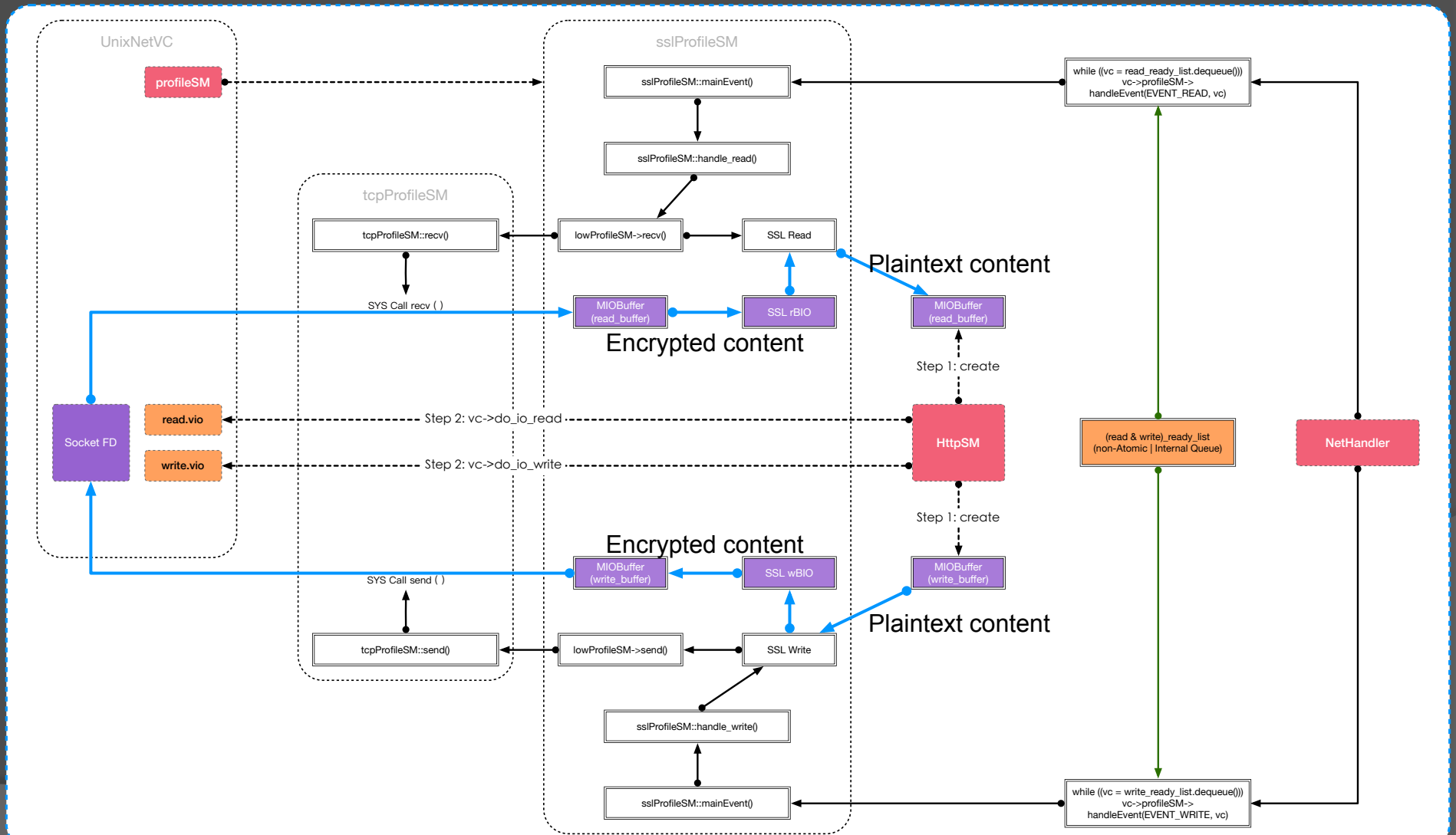
ProfileSM

- In the life of NetVConnection, it may has multiple stages: new born, pre-transfer, transfer, pre-close.
  - For TCP
    - New born: none, Pre-transfer: none,
    - Transfer: read or write, Pre-close: linger close
  - For SSL
    - New born: initial CTX, Pre-transfer: handshake
    - Transfer: encrypt or decrypt, Pre-close: SSL shutdown
- State:
  - New born: do some initialize
  - Pre-transfer: do authorize, ssl handshake
  - Transfer: read/write socket, move data, transform data
  - Pre-close: do linger close, ssl shutdown

ProfileSM - stateful



sslProfileSM – bidirectional



ProfileSM chain

- NetVConnection is a framework for data transmission between socket fd and IOBuffer.
- ProfileSM as the helper to define how to
  - Send/receive data
  - Transform data if necessary
- Like Transform Plugin, ProfileSM can be chained one by one, but it is bidirectional.

Socket FD ←→ tcpProfileSM ←→ sslProfileSM ←→ HttpSM

NetVC + ProfileSM

- NetVC + tcpProfileSM = UnixNetVC
- NetVC + tcpProfileSM + sslProfileSM = SSLNetVC
- NetVC + tcpProfileSM + sslProfileSM + sslProfileSM = SSL in SSL Tunnel
- NetVC + udpProfileSM = UDPNetVC
- NetVC + udpProfileSM + dtlsProfileSM = DTLSNetVC

Based on 6.0.x branch

ATS INTERNALS

Oct 2020 Updated

Agenda

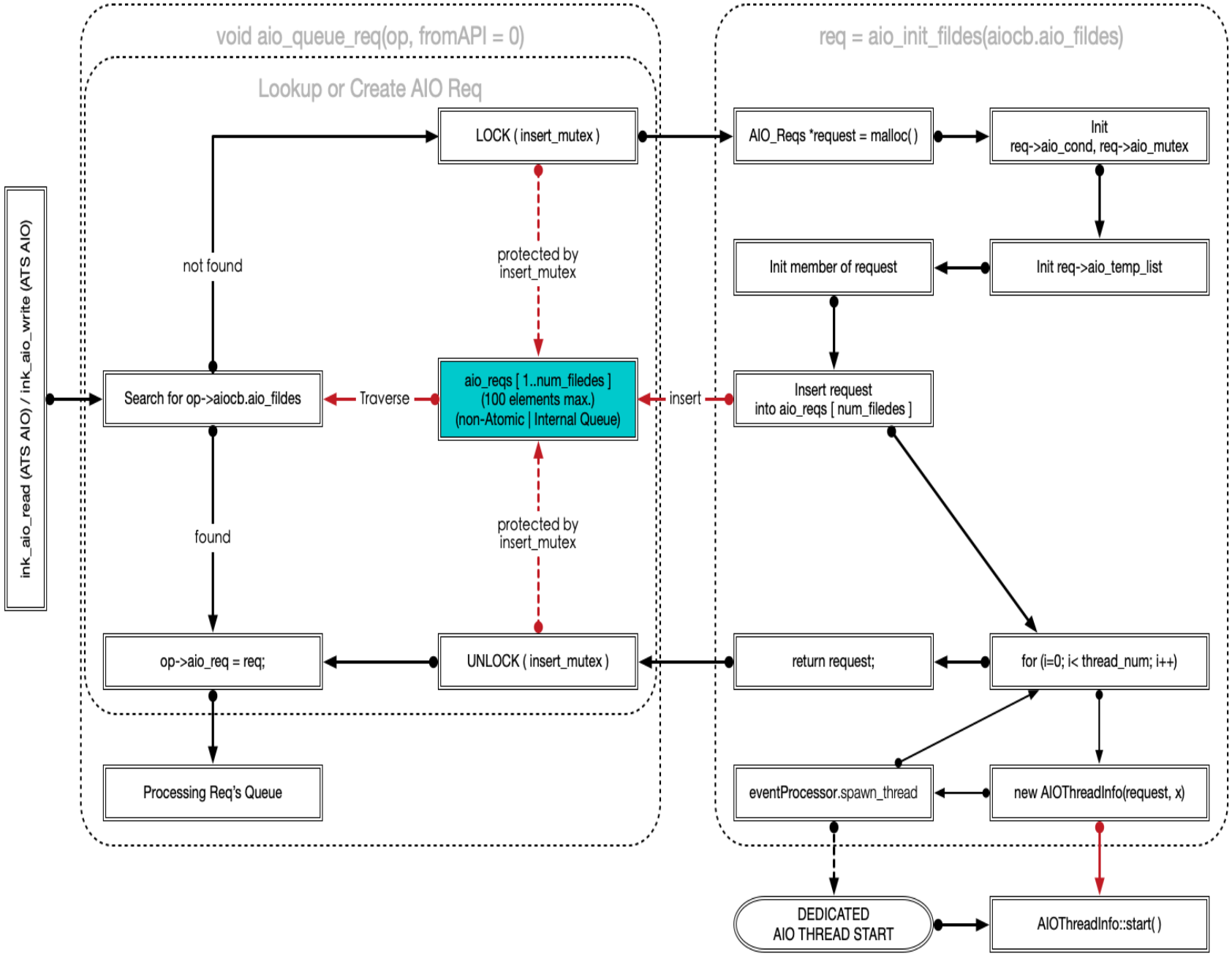
- AIO Sub-system & Native AIO
- DNS Sub-system
- TransformVConnection

AIO Sub-system

AIO Sub-system

- AIO_Reqs per file description
  - AIO_Reqs *aio_reqs[MAX_DISKS_POSSIBLE]
  - MAX_DISKS_POSSIBLE = 100
  - aio_reqs[ ] is protected by `insert_mutex'
- An AIO_Reqs has
  - 8 AIO threads: proxy.config.cache.threads_per_disk
  - An atomic queue: aio_temp_list
  - A sort (by priority) queue: aio_todo, http_aio_todo (p == 0)
    - Protected by aio_mutex and aio_cond

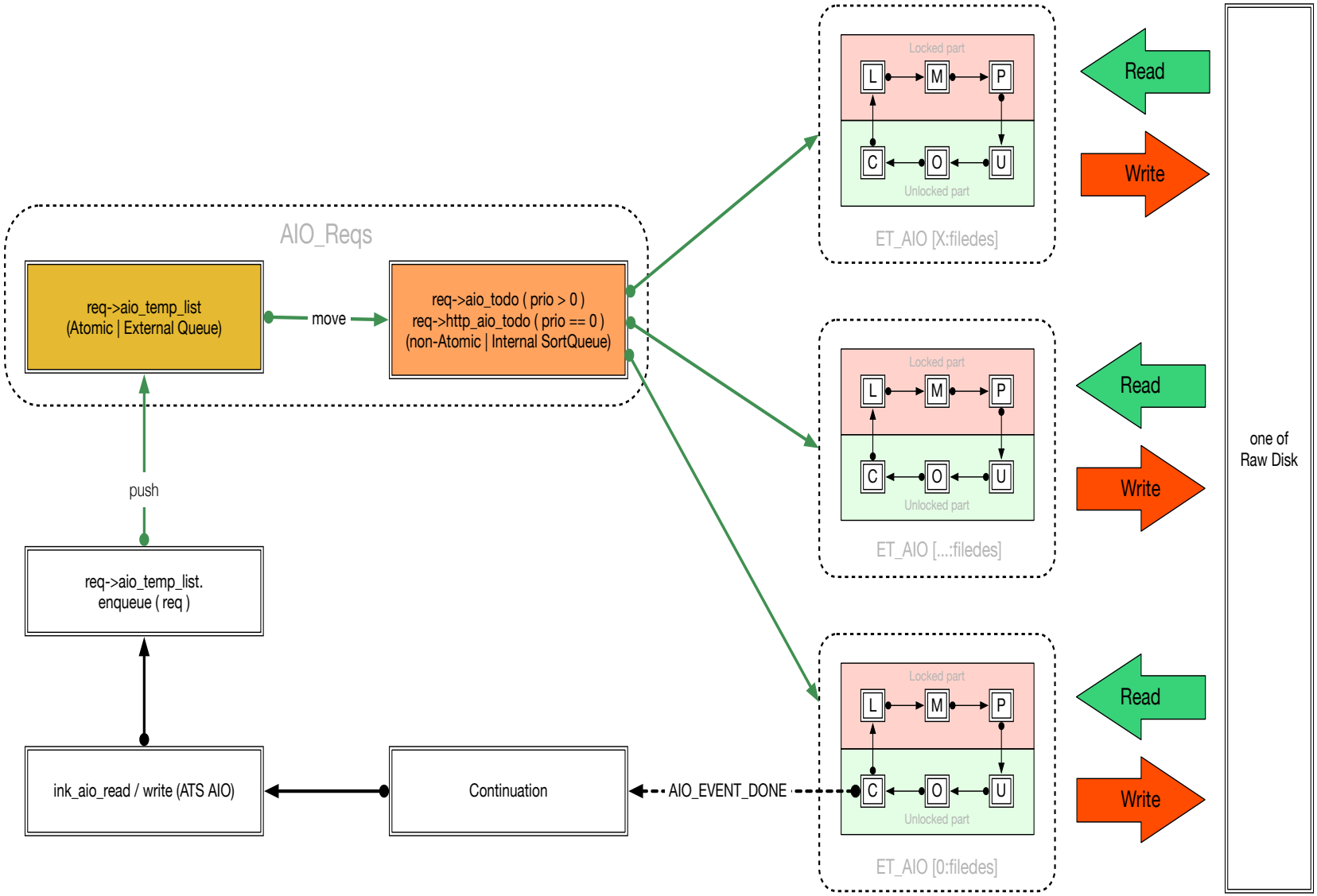
AIO Tasks Queue Lookup, Create and Insert



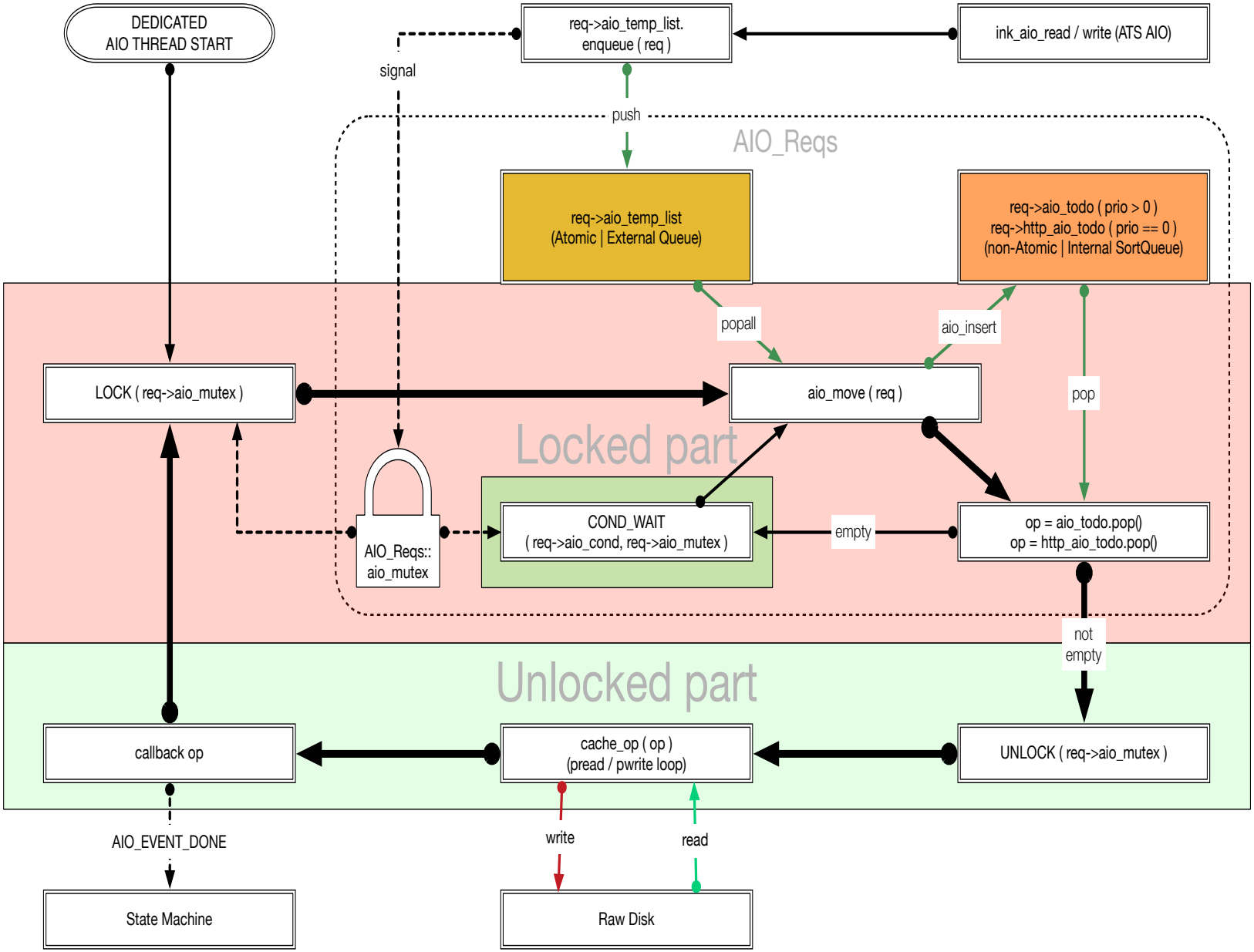
AIO Sub-system (cont.)

- The AIO loop is blocked by one of the following operations
  - Disk I/O operations
  - Conditional wait
- Therefore, each AIO loop consumes only one I/O task from the queue.
- Create multiple AIO Threads to support concurrent I/O operation on specify file description (block device)

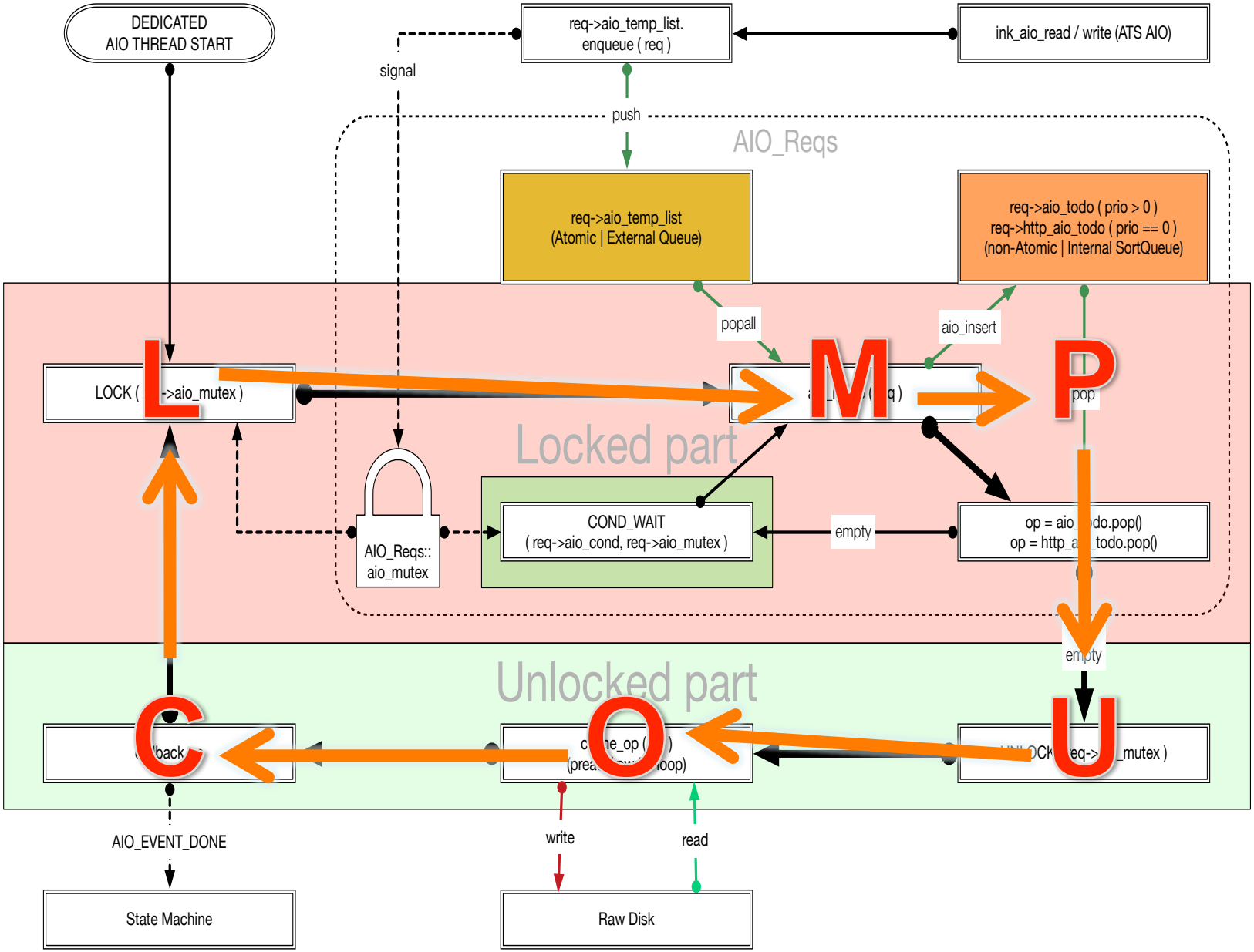
AIO Task Queue and AIO Thread Group



AIO Thread Loops



AIO Thread Loops

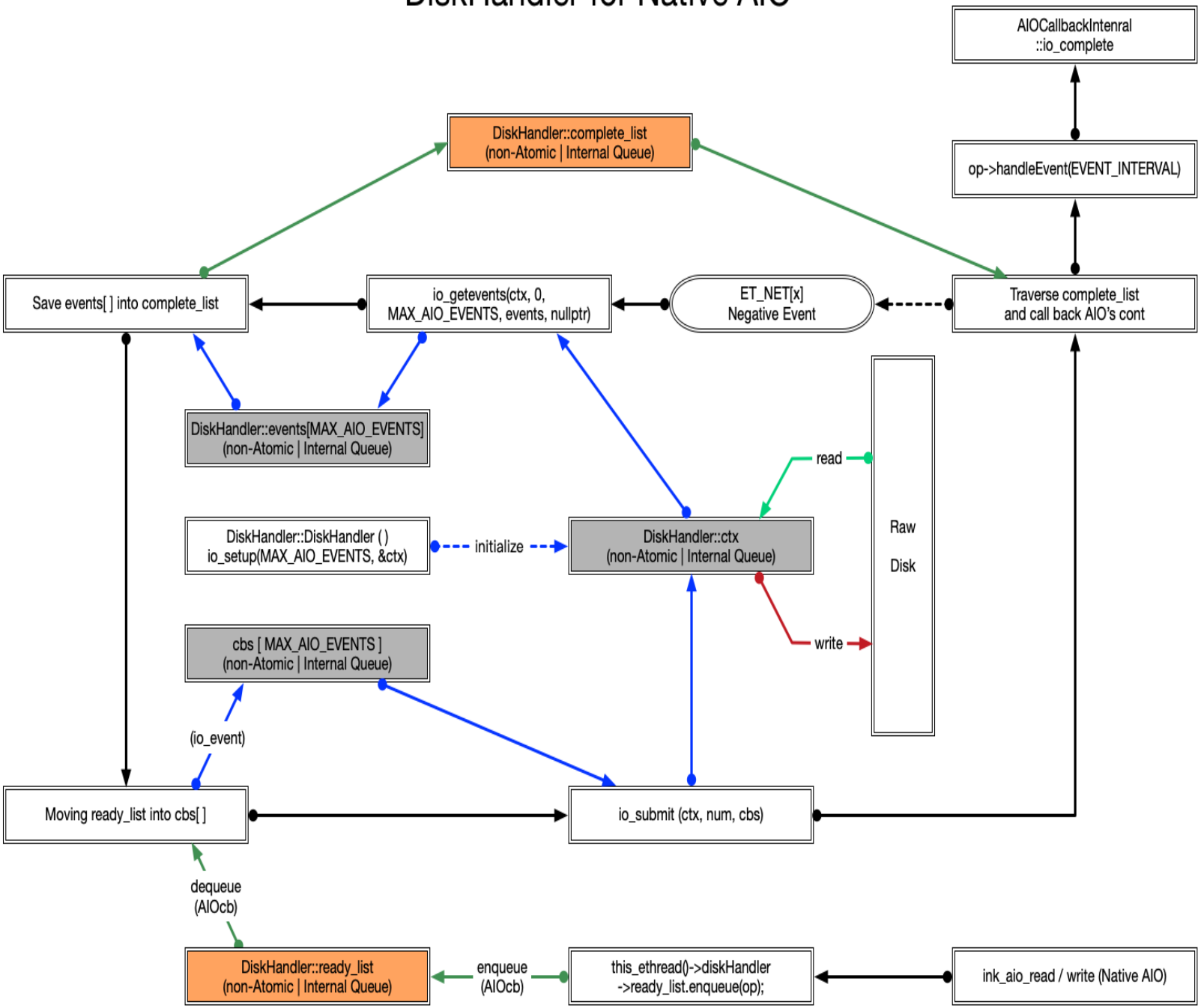


Native AIO

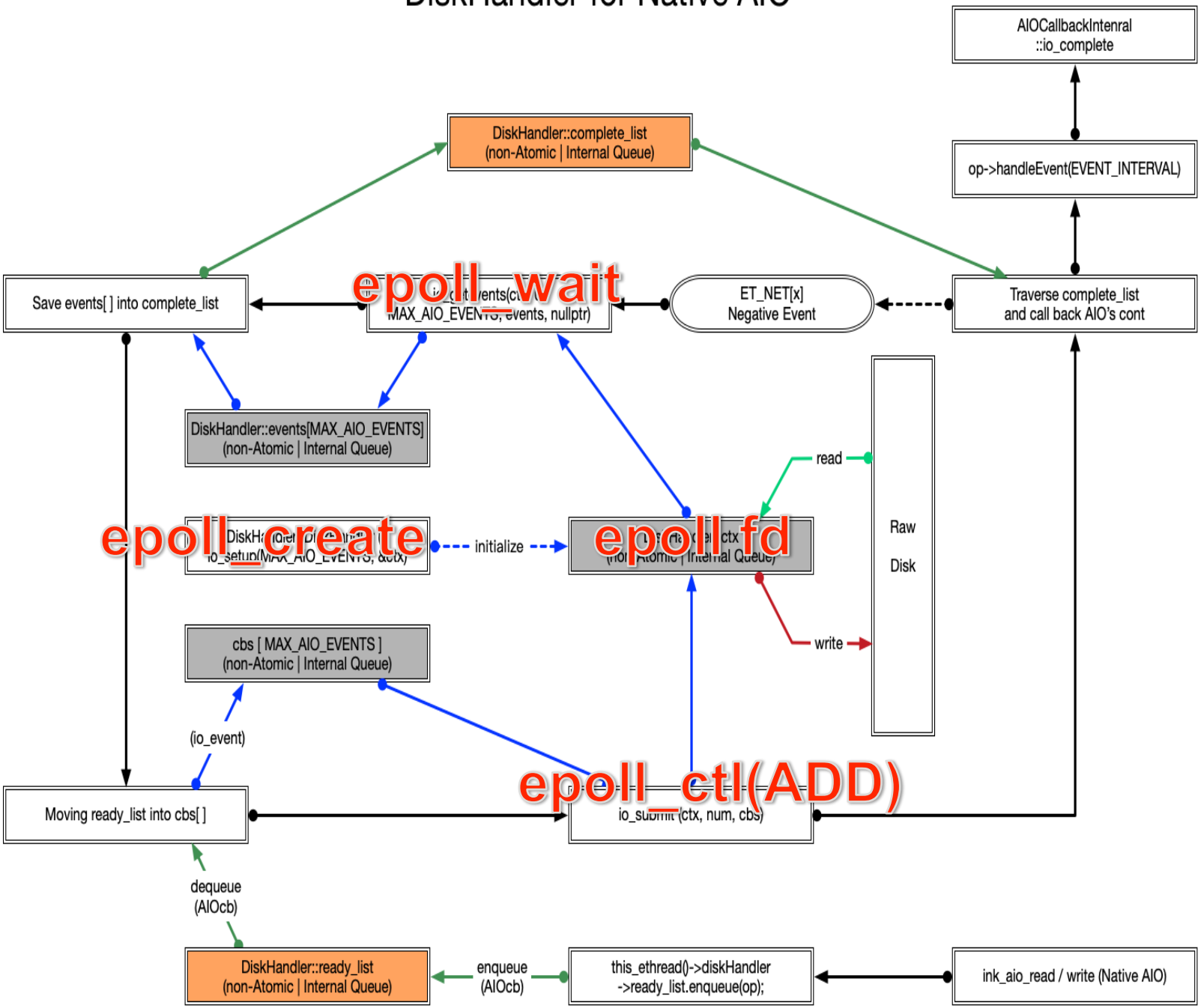
Native AIO

- Native AIO is similar to epoll system
  - io_setup() VS epoll_create()
  - io_getevents() VS epoll_wait()
  - io_submit() VS epoll_ctl()
  - DiskHandler VS NetHandler

DiskHandler for Native AIO



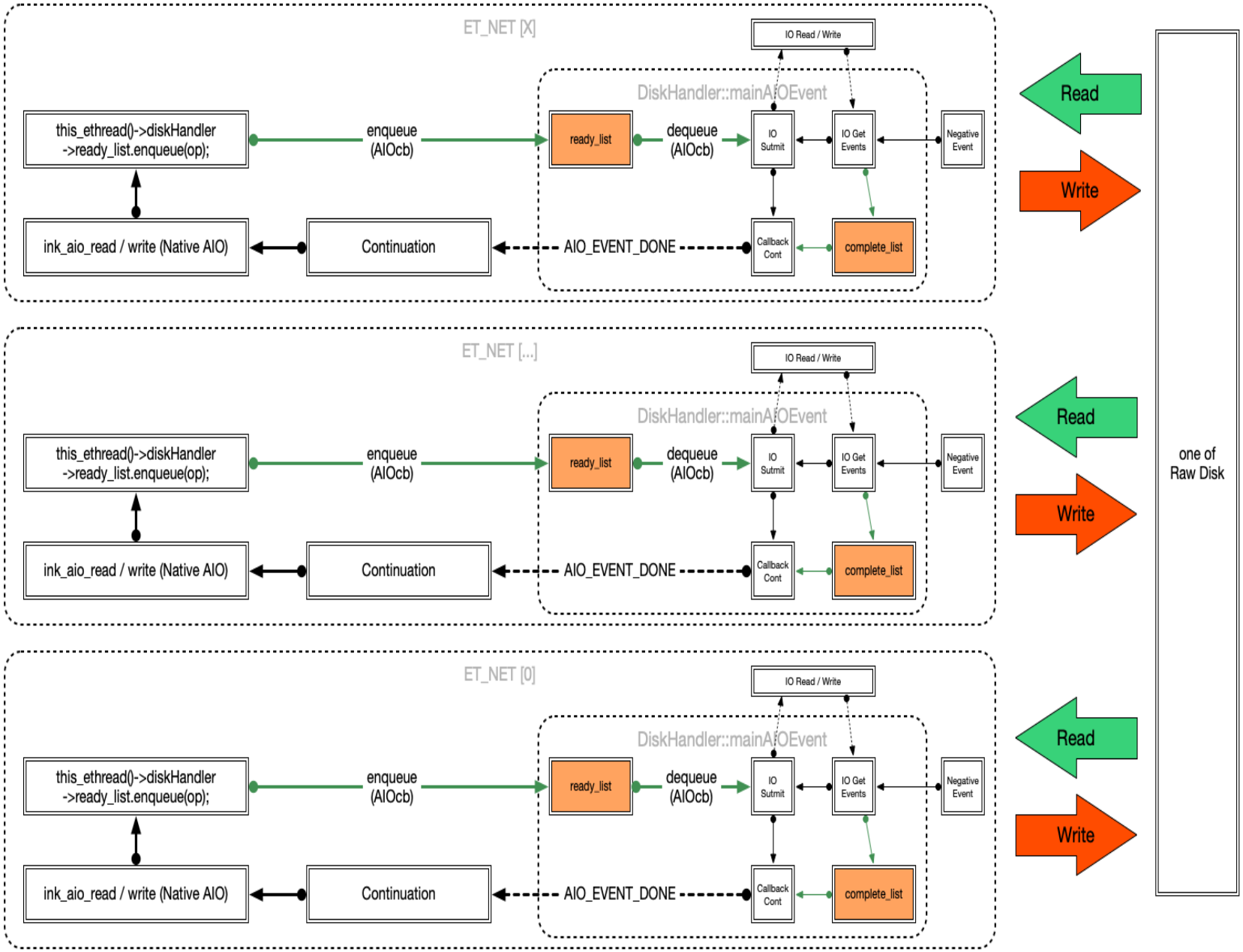
DiskHandler for Native AIO



Native AIO (cont.)

- Similar to NetHandler, DiskHandler is also in every ET_NET threads
  - Different from NetHandler, DiskHandler bundle to its EThread.
  - It shares mutex with its EThread.
- I/O tasks queue : DiskHandler::ready_list
  - EThread local queue
  - Access from current EThread only
- The level of concurrent I/O operations is controlled internally by Native AIO.

Native AIO Task Queue and DiskHandler



DNS Sub-system

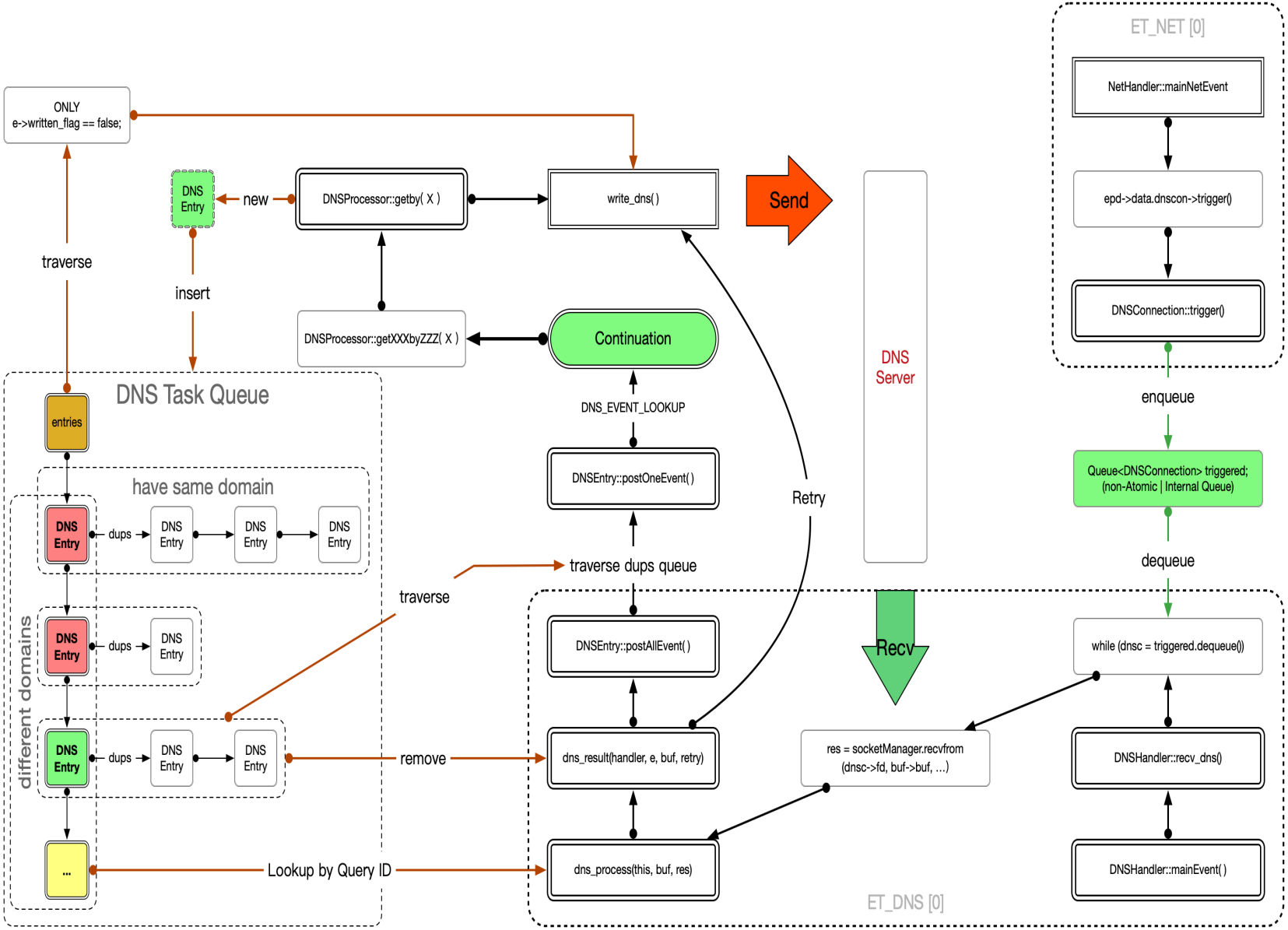
DNS Sub-system

- DNS Task Queue : DNSHandler::entries
  - Pending Entry
    - New queries / retry queries
    - Create an unique Query ID for each DNS request
    - Once DNS requests send out, It will be set to “In flight”
    - Lookup them by domain name
  - In flight Entry
    - DNS requests waiting for response
    - Lookup by domain name / Query ID

DNS Sub-system (cont.)

- Duplicate Queue : DNSEntry::dups
  - Share DNS results for tasks which lookup for the same domain name.
  - Save duplicate tasks.
  - Traverse the dups queue and call back continuation one by one.

DNS Task Queue and DNSHandler



DNS Sub-system (cont.)

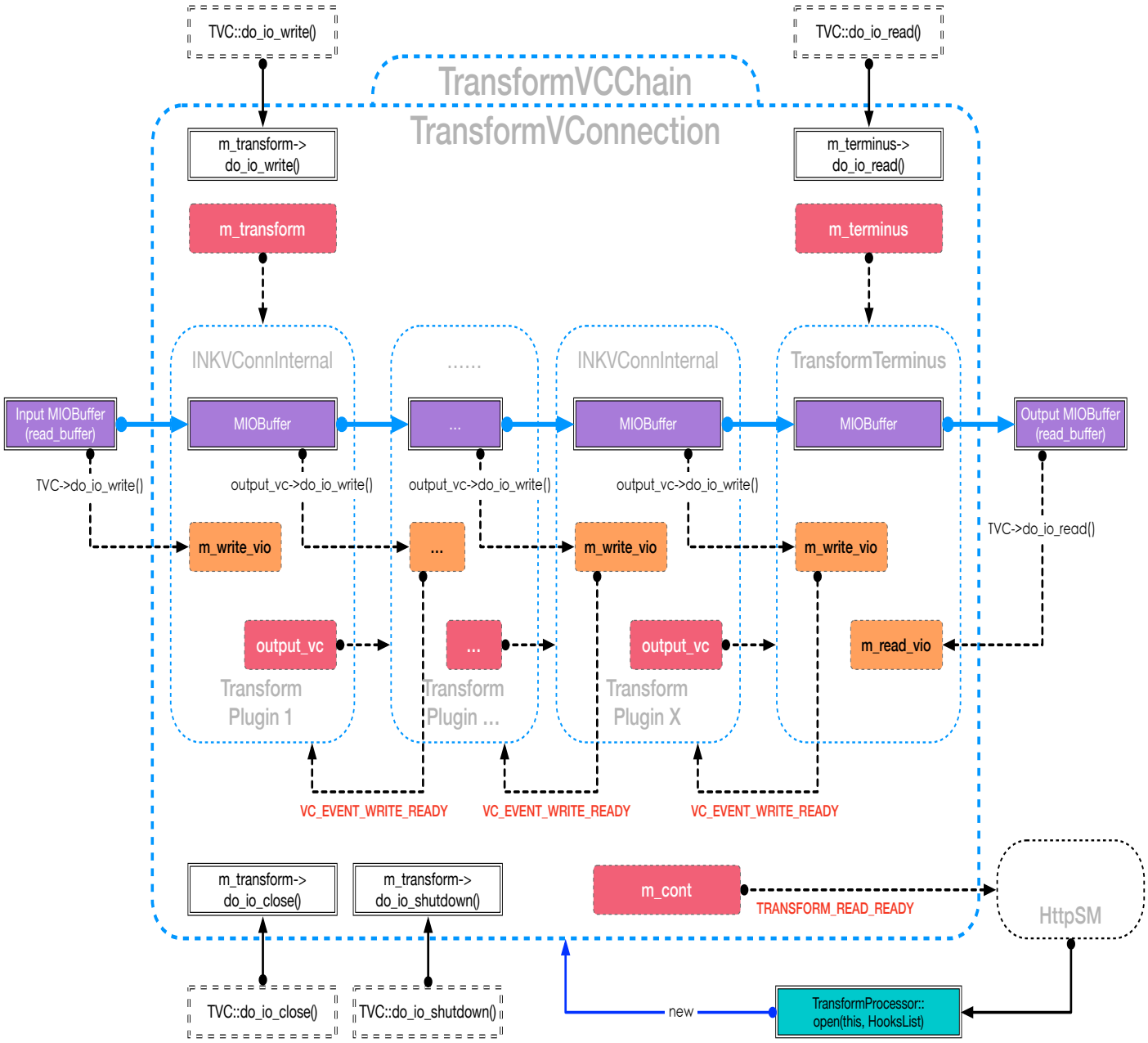
- ET_DNS Thread Group
  - Only one EThread: ET_DNS[0]
  - 2 Key components: DNSHandler and NetHandler
  - If ET_DNS[0] shares EThread with ET_NET[0]
    - There is DNSHandler in ET_NET[0]
  - DNSHandler bundle to ET_DNS[0]
    - It shares mutex with EThread.
- Polling on DNSConnection (UDP socket fd)
  - Only on ET_DNS[0] or ET_NET[0]
  - DNSConnection con[MAX_NAMED], MAX_NAMED = 32

TransformVConnection

TransformVConnection

- What is TransformVConnection ?
  - TVC is a pipe/chain that is connected by one or more INKVConnInternals.
- TVC is a unidirectional pipe
  - The 1st INKVConnInternal is the input
  - The TransformTerminus is always attached to the tail as the output
  - When the TransformTerminus received any data from its upstream, it will send TRANSFORM_READ_READY event to the owner of TVC.

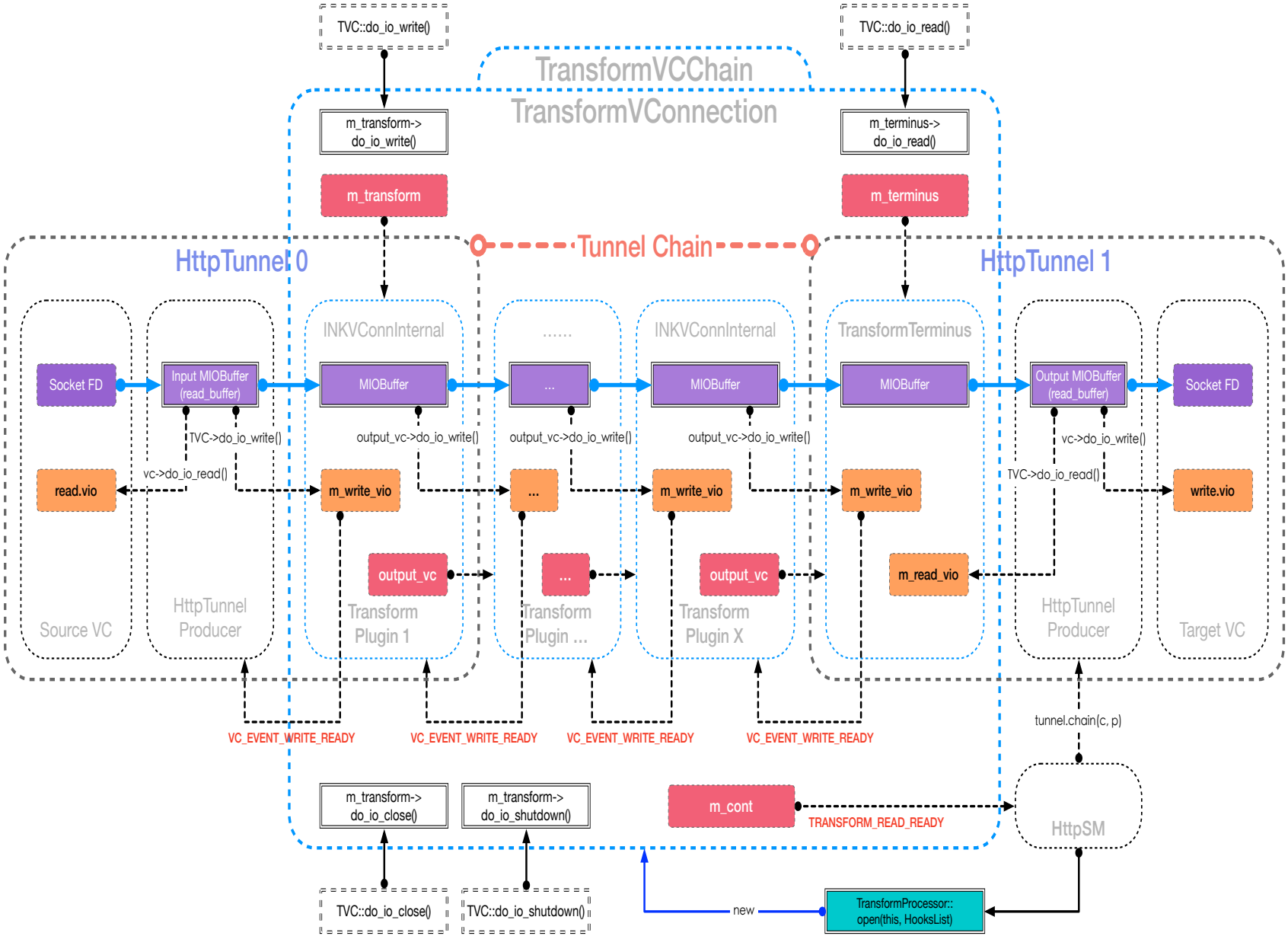
TransformVConnection



TransformVC and Tunnel Chain

- TVC as both a consumer and a producer, connecting two tunnels
- The two tunnels are chained in order to drive the data stream from TVC's input to output
- It is a complete pipe from the source VC to the target VC (for example: from client VC to server VC).

TransformVConnection and Tunnel Chain



Req and Resp Transform

- Request Transform
  - For POST and PUT methods, accept chunked and non-chunked content.
  - The 1st HttpTunnel only verifies the integrity of the chunks but not de-chunk them. Therefore, HttpTunnel sends raw chunked content to the TransformVC.
  - The plugin should identify the encoding type and decode chunked content by itself. The plugin should not change the encoding type.
  - The downstream requires the exact length of content, which means the plugin should collect all chunks and get the length of raw content before write any data to downstream if the incoming content is encoded in chunked.
  - It is easy to get the exact length of content if the plugin could get the value of `Content-Length` from the request.

Req and Resp Transform (cont.)

- Response Transform
  - For any request which has a payload with response, accept chunked and non-chunked content.
  - The 1st HttpTunnel identify the encoding type and decode chunked content automatically. Therefore, HttpTunnel always sends de-chunked content to the TransformVC.
  - The plugins always receives de-chunked content and sends de-chunked content to downstream.
  - The INT64_MAX can be the length of content which means the content length is currently uncertain. The plugin should update `write_vio.nbytes` with the exact length if all content is collected.
  - The 2nd HttpTunnel will chunking the content automatically according to the capability of the user agent.

Thanks