Hunter/Trinity Migration

Merging Hunter and Trinity into a new product

Gene Gershanok

VP Engineering

Dec 5, 2017

Vidyo®

Goals

What we're trying to achieve

- Combine the years of solid features of Hunter with the scale of Trinity to create a unified architecture.
  - Preserve Hunter query based features
    - User Search, Roster, Authentication etc
  - Leverage Trinity Real-Time features
    - Highly scalable XMPP server designed for persistent connections/notifications
    - Built-in Features
      - Presence/Roster
      - Chat
      - Disco
      - Component connections

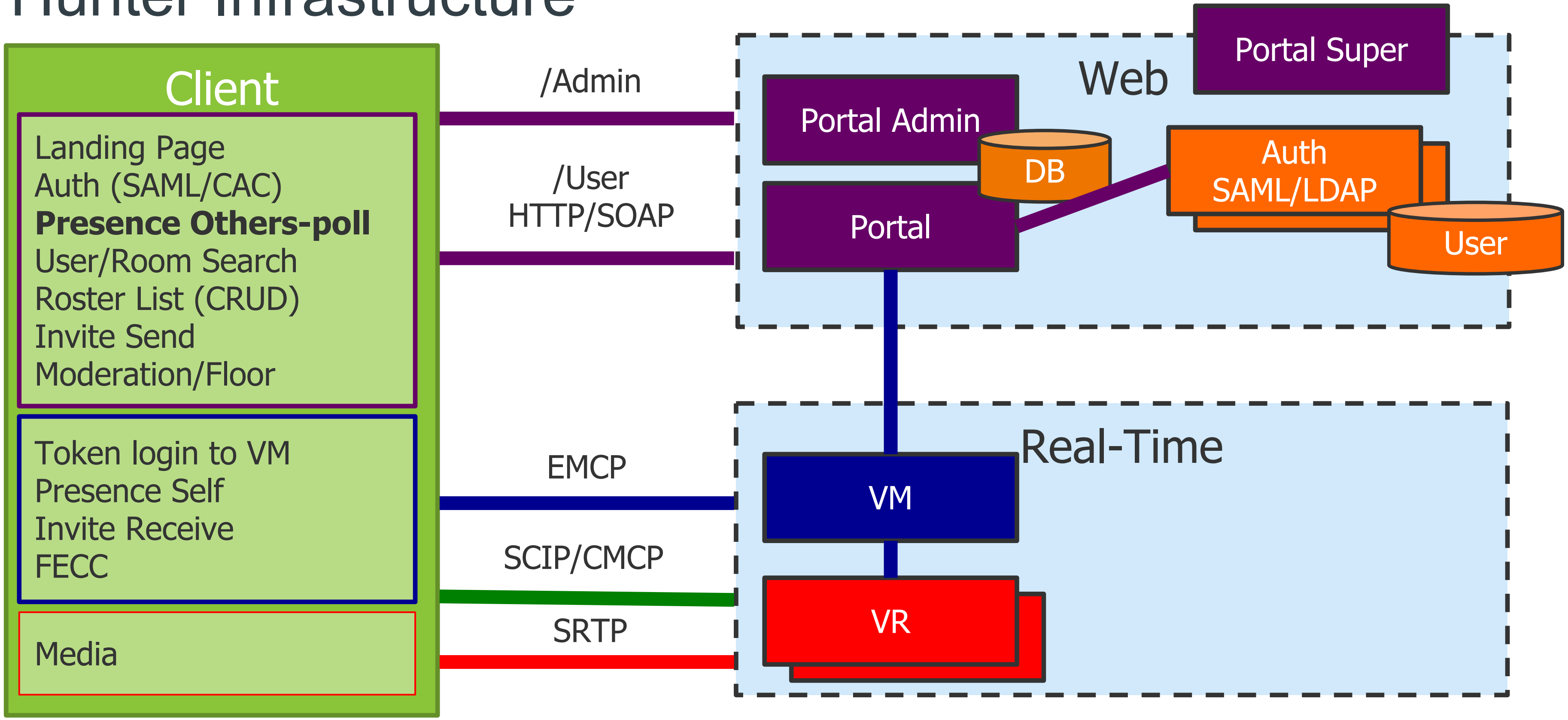
Hunter

Existing architecture for VidyoCloud

- Benefits
  - Numerous “tenant” features (landing pages)
  - Complex login work-flow support/configuration (SAML/LDAP)
  - GUI Super/User
- Drawbacks
  - Monolithic (hard to scale)
  - Everything goes through the DataBase
  - Using real-time connectivity with Web technology, ends up in polling etc
  - Sharing state between VM and Portal (presence, invite, etc)

Vidyo®

Hunter Infrastructure



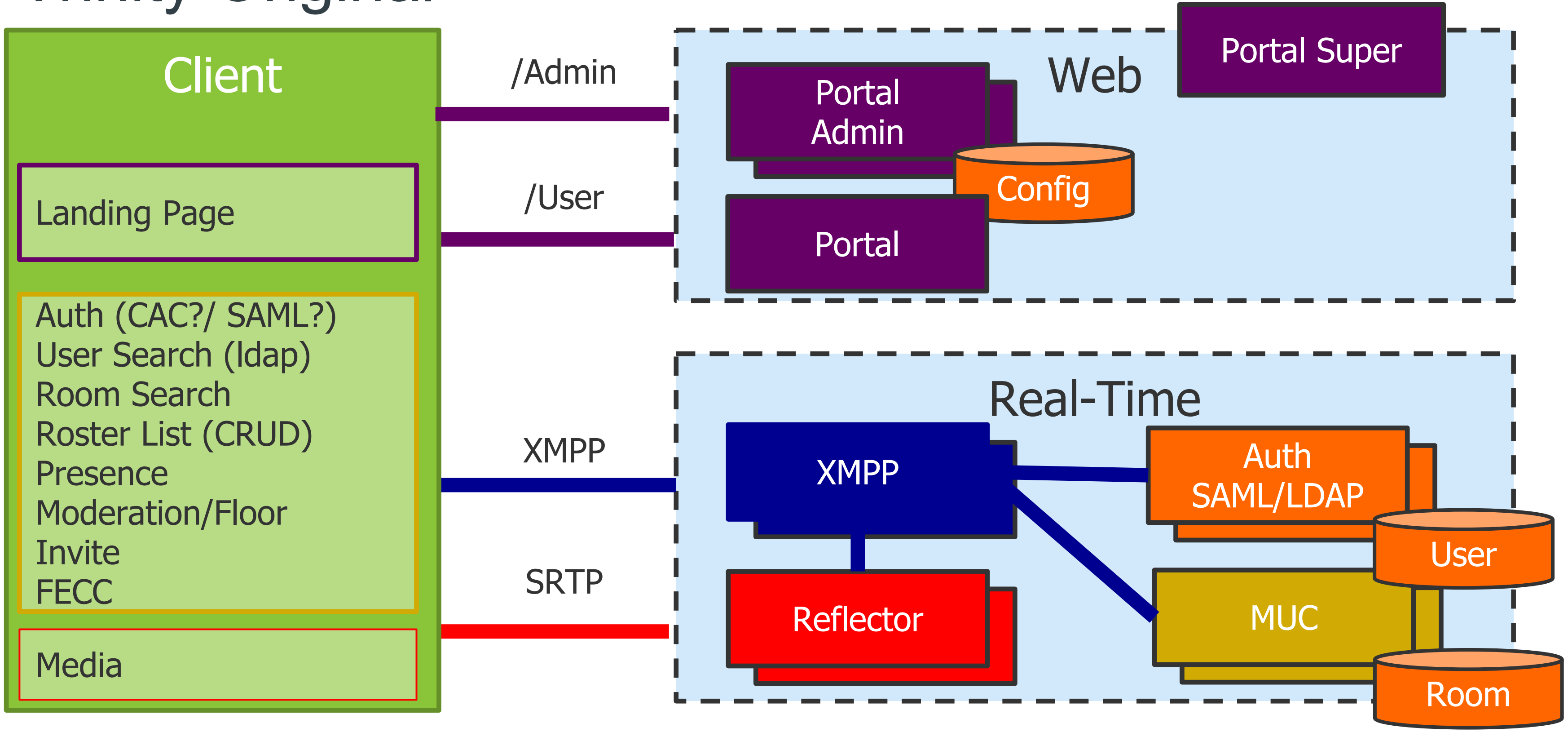
Vidyo®

Trinity Original

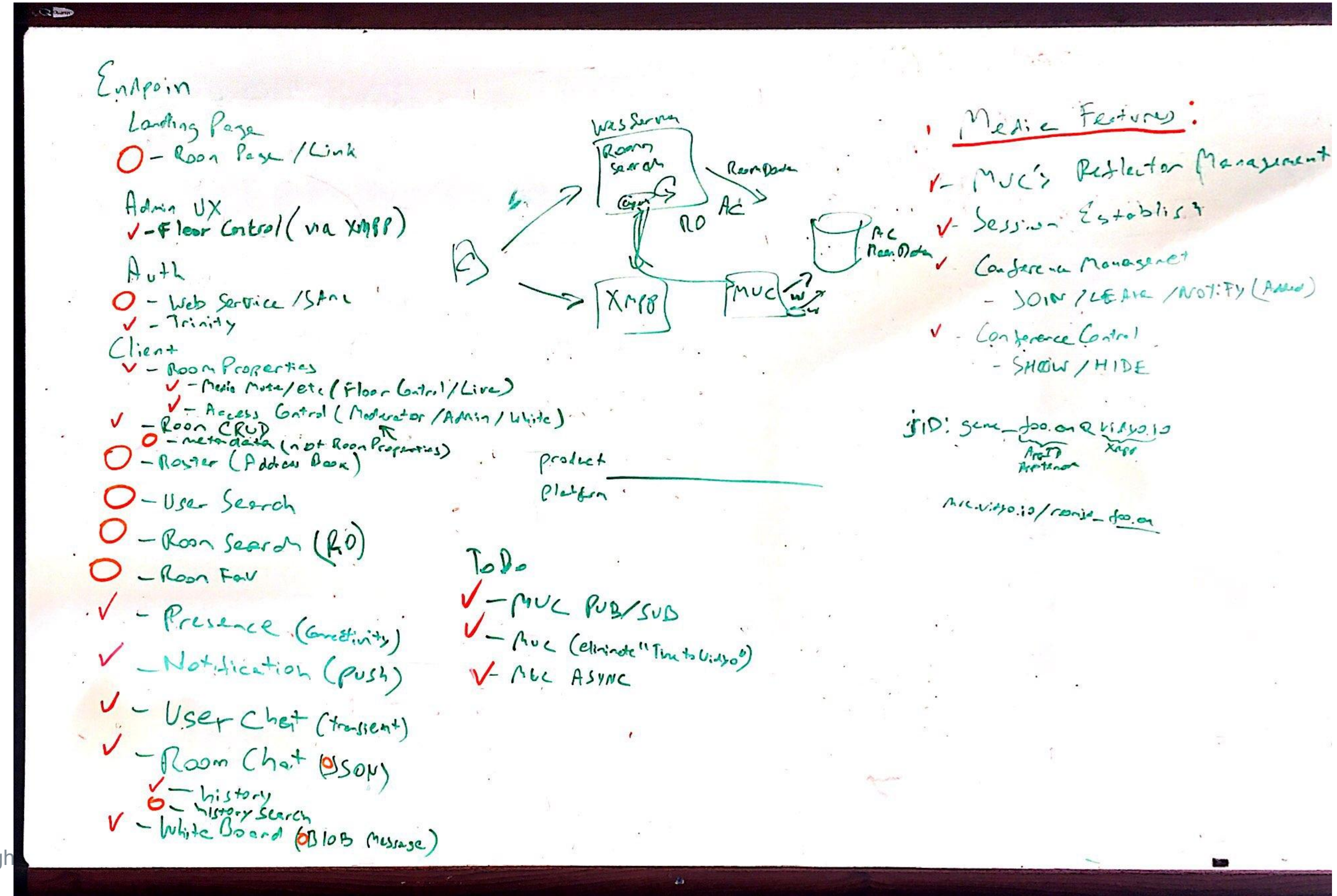
Existing architecture partly used for Vidyo.io

- Benefits
  - Designed for scale of real-time connections
  - Leverage existing MongooseIM XMPP server
  - Single pin-hole for persistent connections
  - Components are written in C/erlang for performance
- Drawbacks
  - Currently does not have advanced authentication implemented for 1000+ tenants (LDAP)
  - GUI for management of Tenants (AUTH/etc)
  - Hard transition for customers needing REST APIs
  - Components are written in C for non real time as well (access to DB)
  - Everything plumbed through SDK
  - Available Frameworks not as mainstream (spring security, DB access) etc

Trinity Original



Agreed upon Architecture



Endpoin

Landing Page
- O – Room Page / Link

Admin UX
- ✓ – Floor Control (via XMPP)

Auth
- O – Web Service / SAML
- ✓ – Trinity

Client
- ✓ – Room Properties
  - ✓ – Media Mute/etc (Floor Control/Live)
  - ✓ – Access Control (Moderator / Admin / White)
- ✓ – Room CRUD
  - O – metadata (not Room Properties)
- O – Roster (Address Book)
- O – User Search
- O – Room Search (RO)
- O – Room Fav
- ✓ – Presence (Connectivity)
- ✓ – Notification (push)
- ✓ – User Chat (transient)
- ✓ – Room Chat (O JSON)
  - ✓ – history
  - O – history search
- ✓ – White Board (O BLOB Message)

ToDo
- ✓ – MUC PUB/SUB
- ✓ – MUC (eliminate "Time to Vidyo")
- ✓ – MUC ASYNC

Media Features:
- ✓ – MUC's Reflector Management
- ✓ – Session Establish
- ✓ – Conference Management
  - – JOIN / LEAVE / NOTIFY (Added)
- ✓ – Conference Control
  - – SHOW / HIDE

JID: $\text{some\_}\underbrace{\text{foo.on}}_{\text{AppID / AppTenant}}\text{@}\underbrace{\text{vidyo.io}}_{\text{Xmpp}}$

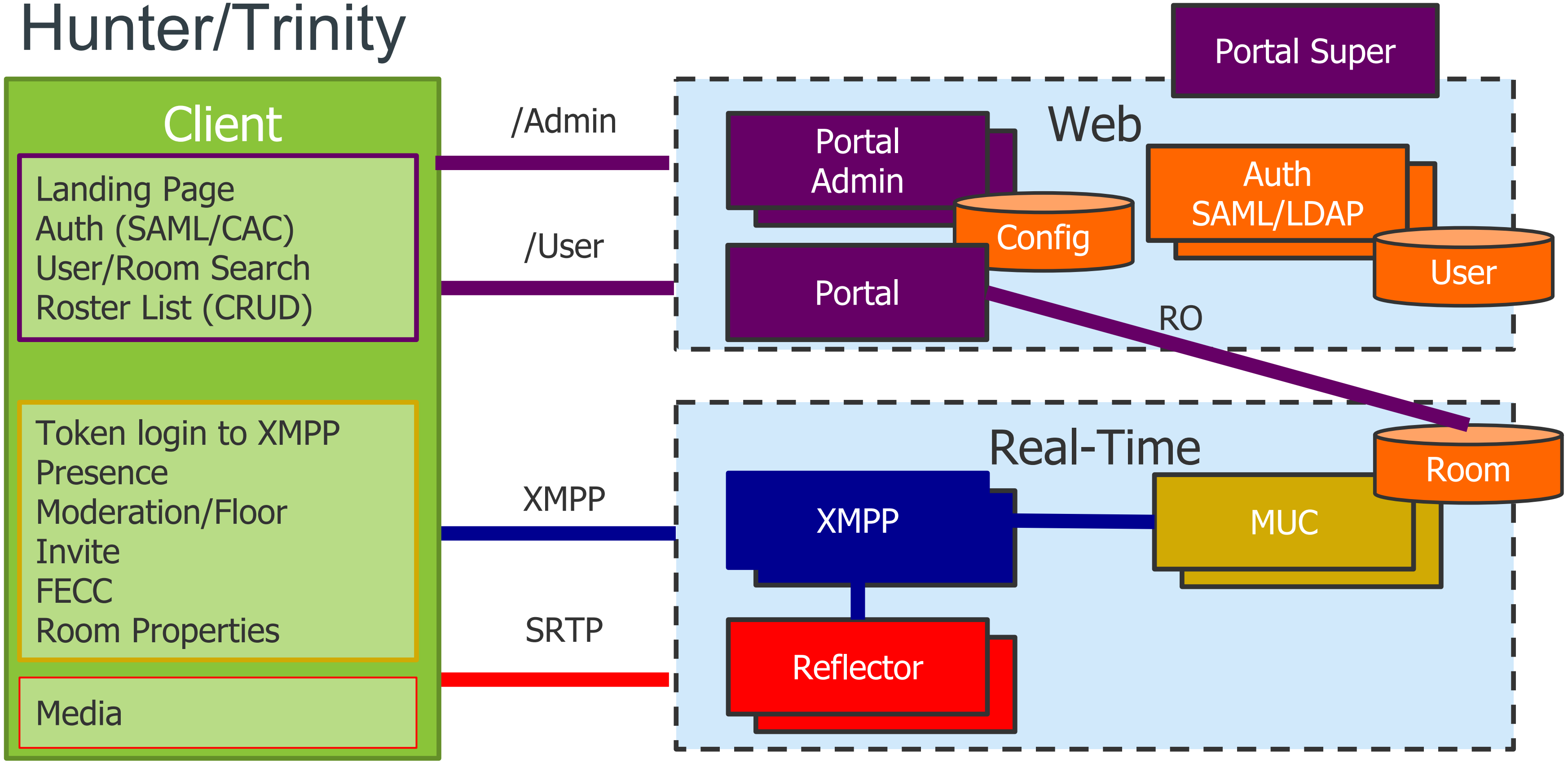
mc.vidyo.io/roomid_foo.on

Hunter/Trinity Merge

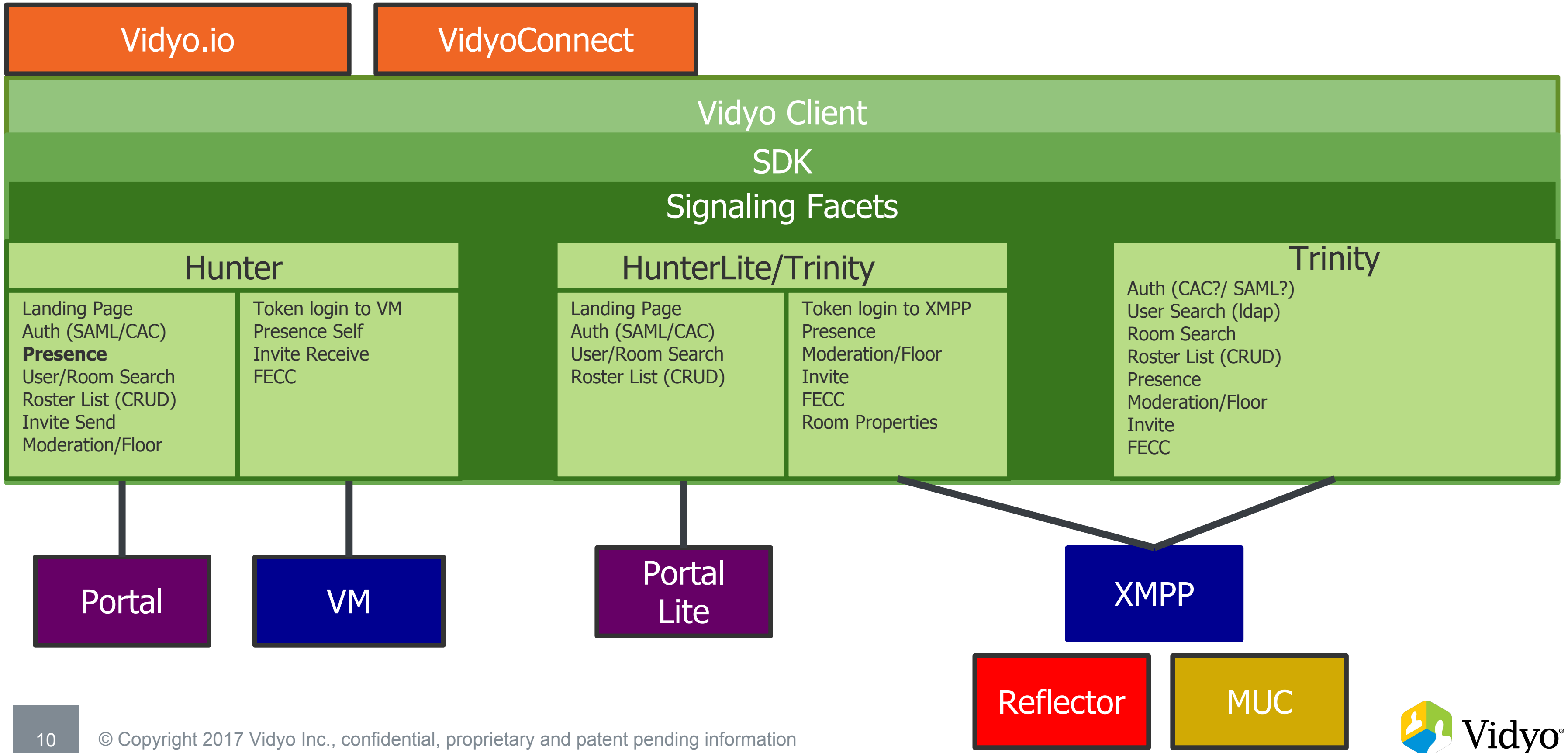
Using the best of both

- Based on the agreed upon architecture (previous slide)
  - Everything with **O** will be re-used from Hunter, ✔ will be from Trinity
  - The big difference is that in the 1st phase **O** will be build by combining Signaling Facets from Neo/Hunter with Trinity to maintain common VidyoClient API during the transition.
- Result
  - Leave DB driven REST based APIs in Hunter (Search, VCard (pic), meta)
  - Leave Real-time async APIs in Trinity (Presence, notification, invite)
  - Leverage years of existing product development
  - Easier to implement Application features using web technology (SAML/Fav Rooms, etc)
  - Call critical features using earlang/C
  - Faster time to market since Hutner had features missing in Trinity, and trinity has scale missing in Hunter
  - Easier transition for customers since they can keep some of Hunter APIs

Hunter/Trinity



VidyoClient



Vidyo®