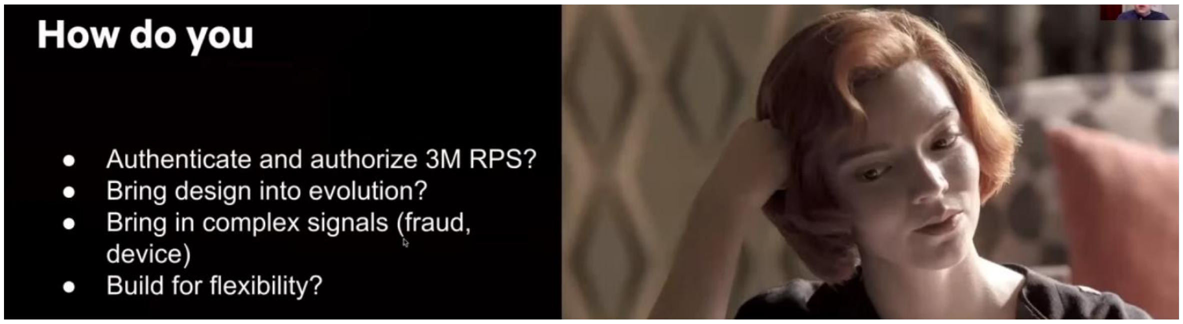
TRACK
Security: Establishing & Maintaining Customer Trust

SESSION

Authorization at Netflix Scale

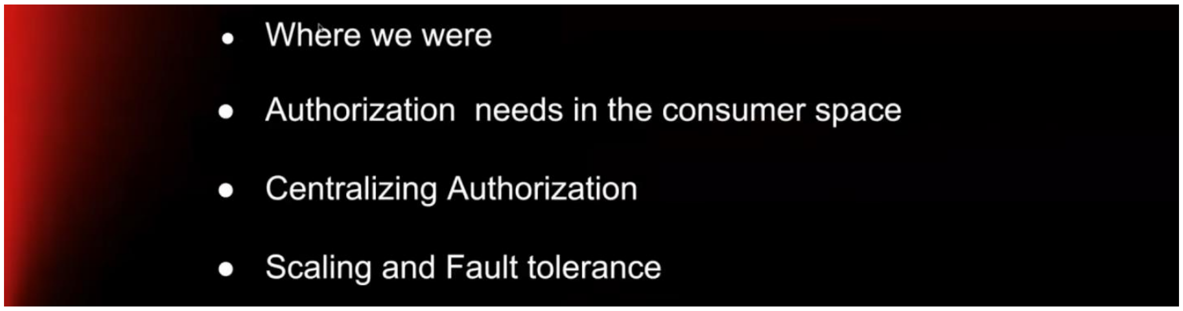
Travis Nelson
Senior Software Engineer @Netflix

Travis Nelson discusses Netflix's approach to scaling and shares techniques for distributed caching and isolating failure domains.

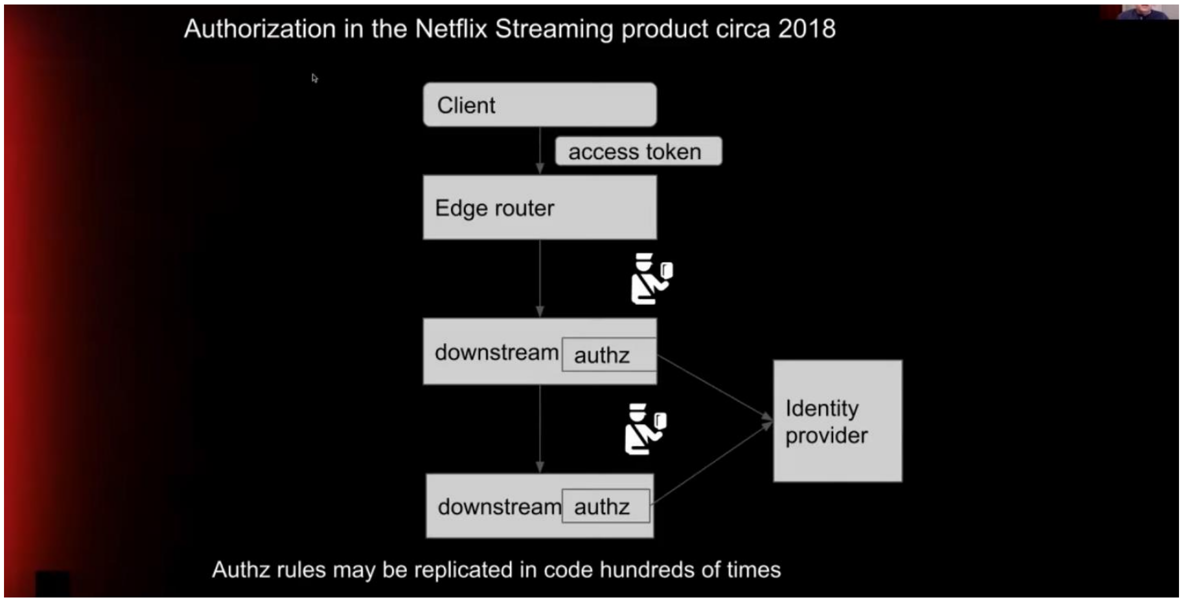


How do you

- Authenticate and authorize 3M RPS?
- Bring design into evolution?
- Bring in complex signals (fraud, device)
- Build for flexibility?

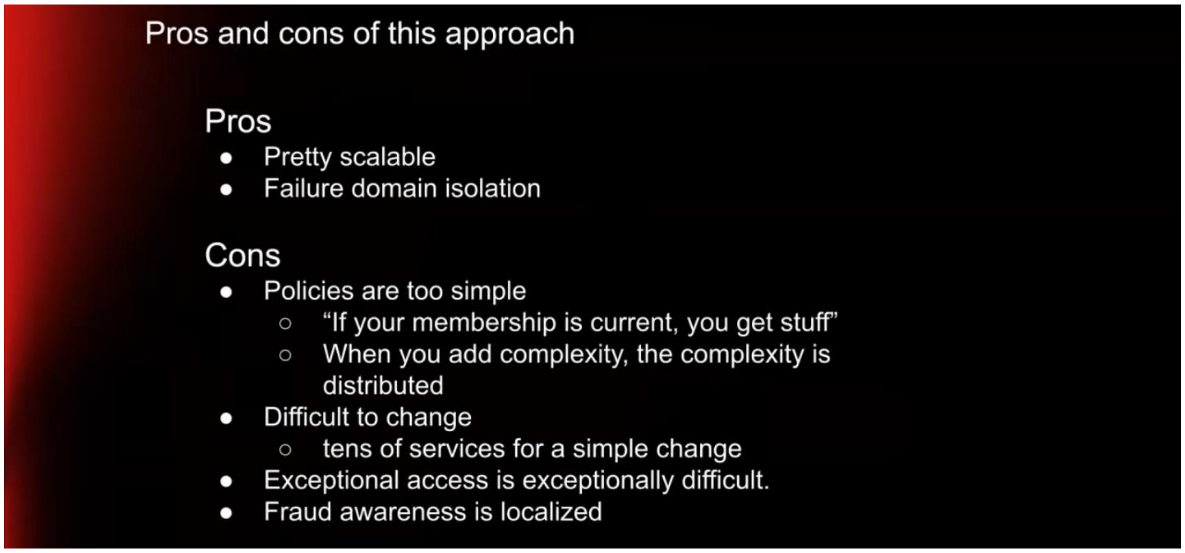


- Where we were
- Authorization needs in the consumer space
- Centralizing Authorization
- Scaling and Fault tolerance



Authorization in the Netflix Streaming product circa 2018

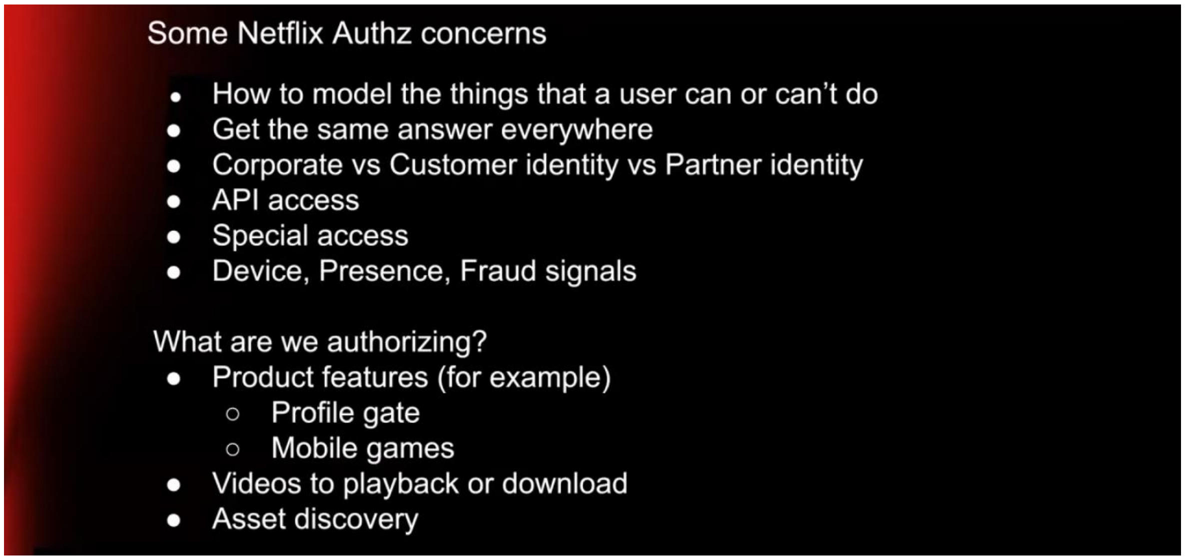
Authz rules may be replicated in code hundreds of times



Pros and cons of this approach

Pros
- Pretty scalable
- Failure domain isolation

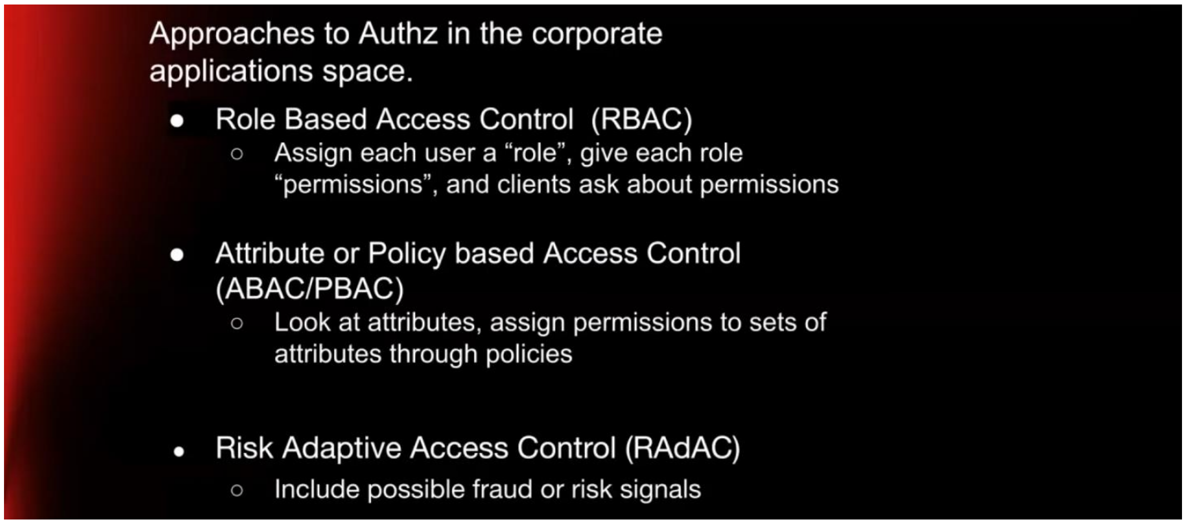
Cons
- Policies are too simple
  - "If your membership is current, you get stuff"
  - When you add complexity, the complexity is distributed
- Difficult to change
  - tens of services for a simple change
- Exceptional access is exceptionally difficult.
- Fraud awareness is localized



Some Netflix Authz concerns

- How to model the things that a user can or can't do
- Get the same answer everywhere
- Corporate vs Customer identity vs Partner identity
- API access
- Special access
- Device, Presence, Fraud signals

What are we authorizing?
- Product features (for example)
  - Profile gate
  - Mobile games
- Videos to playback or download
- Asset discovery

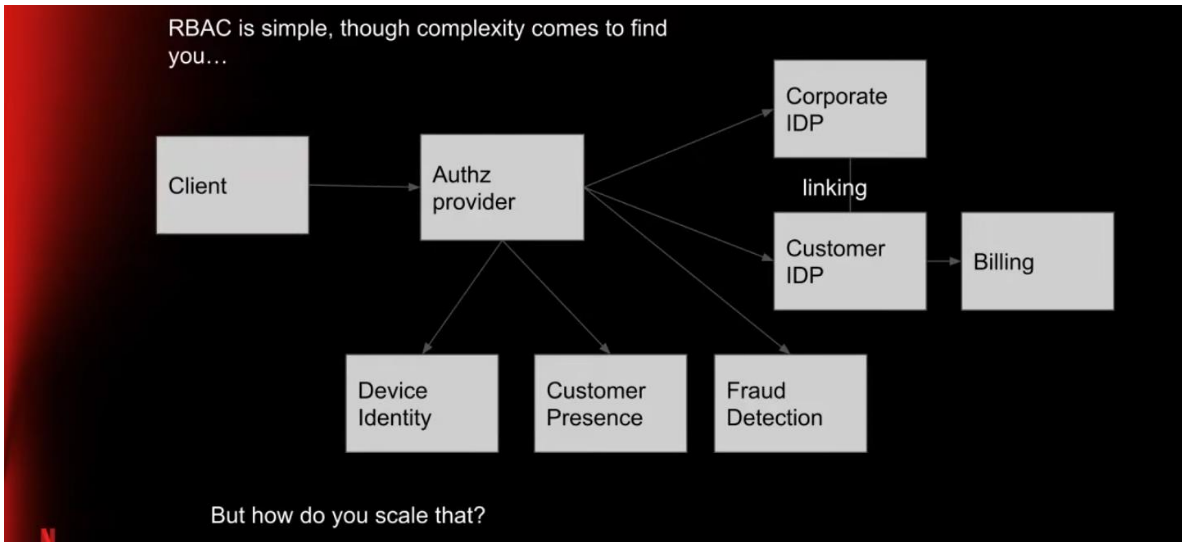


Approaches to Authz in the corporate applications space.

- Role Based Access Control (RBAC)
  - Assign each user a "role", give each role "permissions", and clients ask about permissions

- Attribute or Policy based Access Control (ABAC/PBAC)
  - Look at attributes, assign permissions to sets of attributes through policies

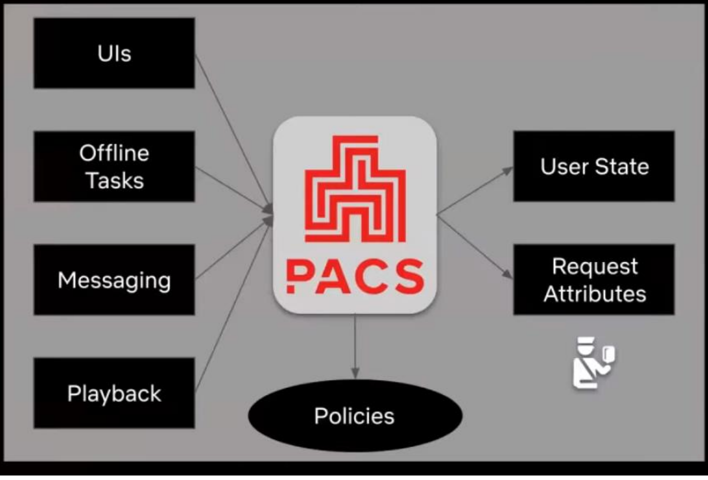
- Risk Adaptive Access Control (RAdAC)
  - Include possible fraud or risk signals



RBAC is simple, though complexity comes to find you…

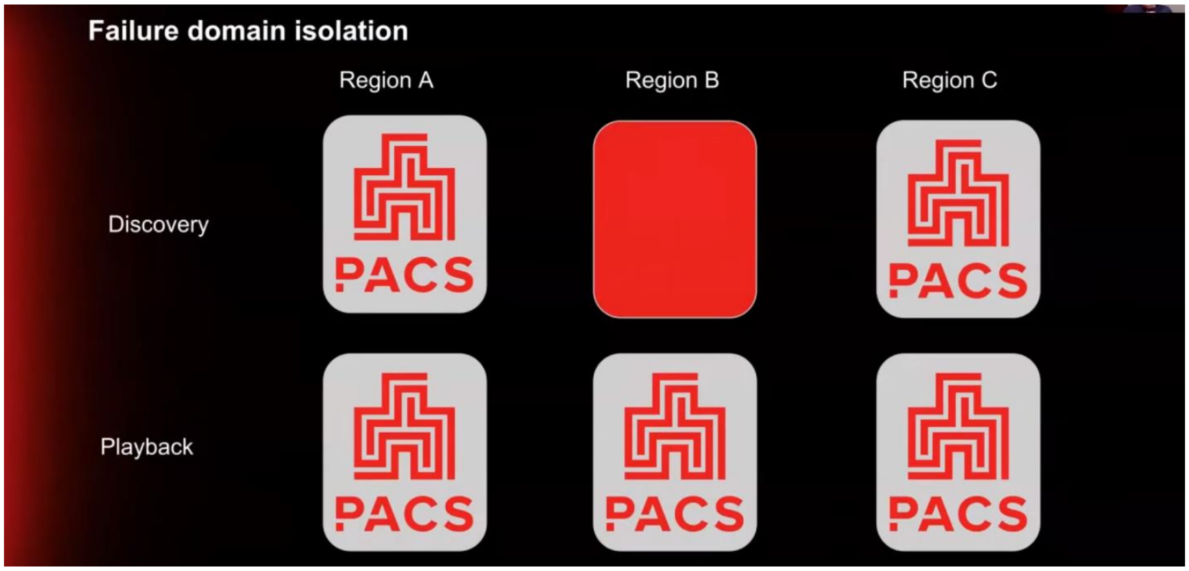
But how do you scale that?

Enter PACS

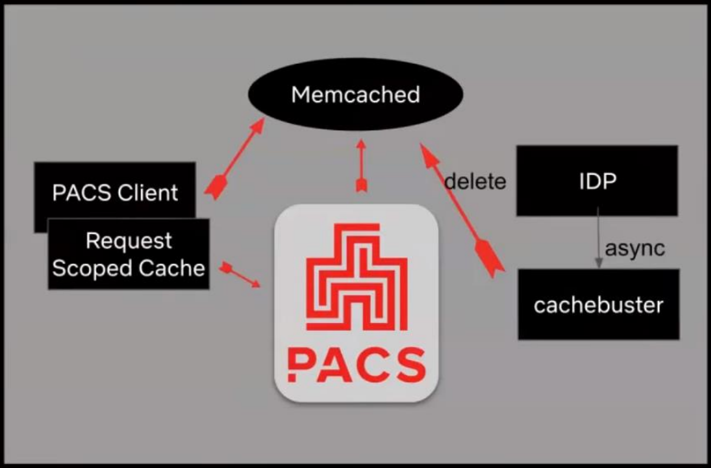


How do we get back failure domain isolation?

Failure domain isolation

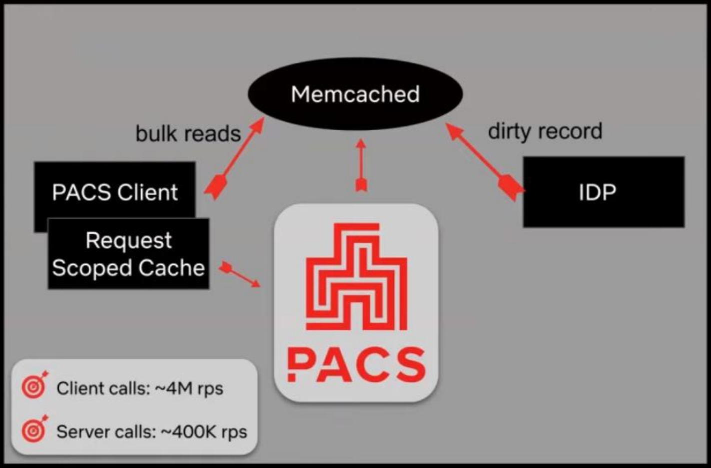


Scaling and cache consistency - approach 1



Latency was too high

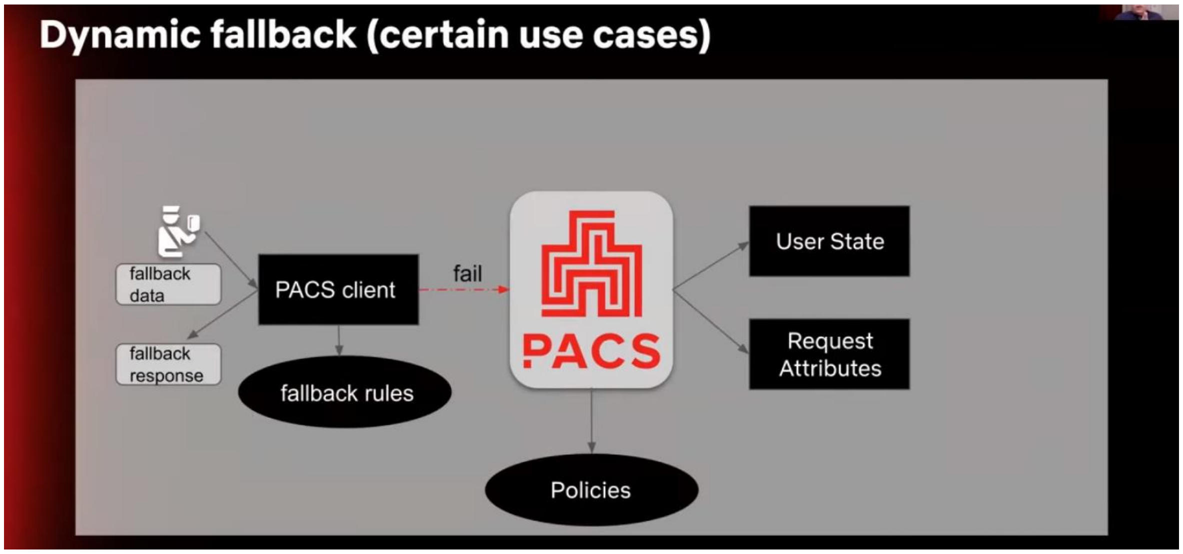
Scaling and cache consistency - approach 2



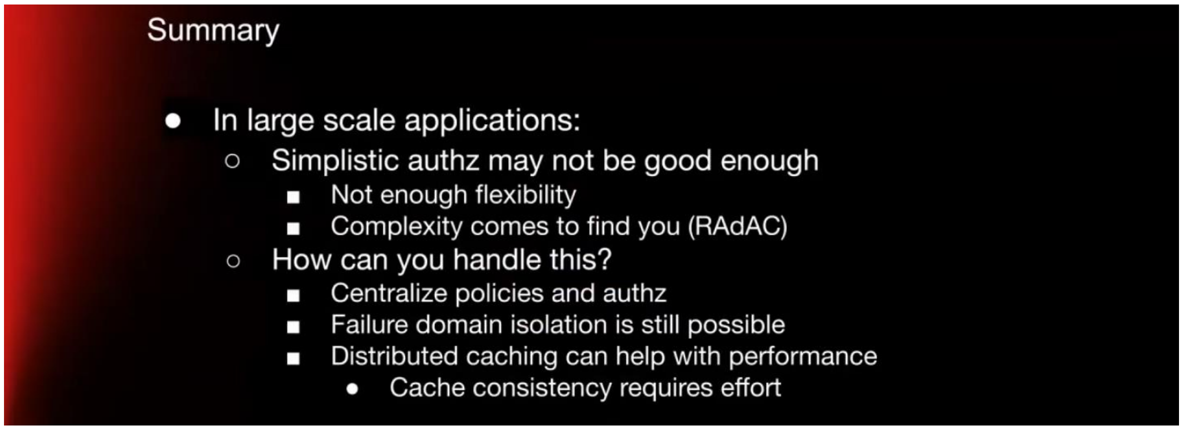
Client calls: ~4M rps

Server calls: ~400K rps

A near-synchronous approach (within a region) was fast enough



Dynamic fallback (certain use cases)



Summary

- In large scale applications:
  - Simplistic authz may not be good enough
    - Not enough flexibility
    - Complexity comes to find you (RAdAC)
  - How can you handle this?
    - Centralize policies and authz
    - Failure domain isolation is still possible
    - Distributed caching can help with performance
      - Cache consistency requires effort