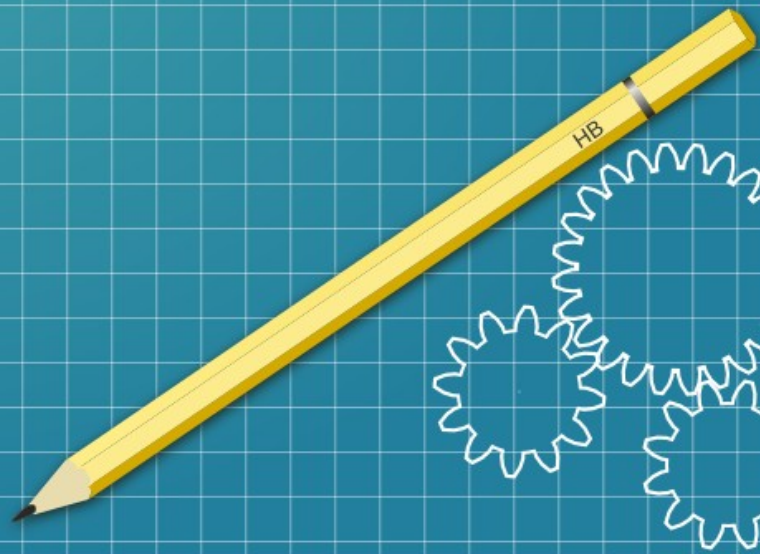
Efficiently handle large numbers of customers

Scalability
Performance
Efficiency
AAA



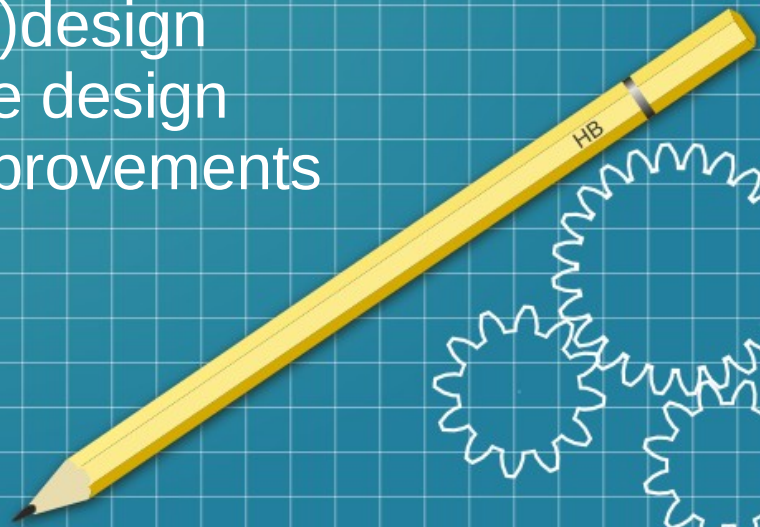
01 Feb 2024

Claudiu Brasovean

ToC

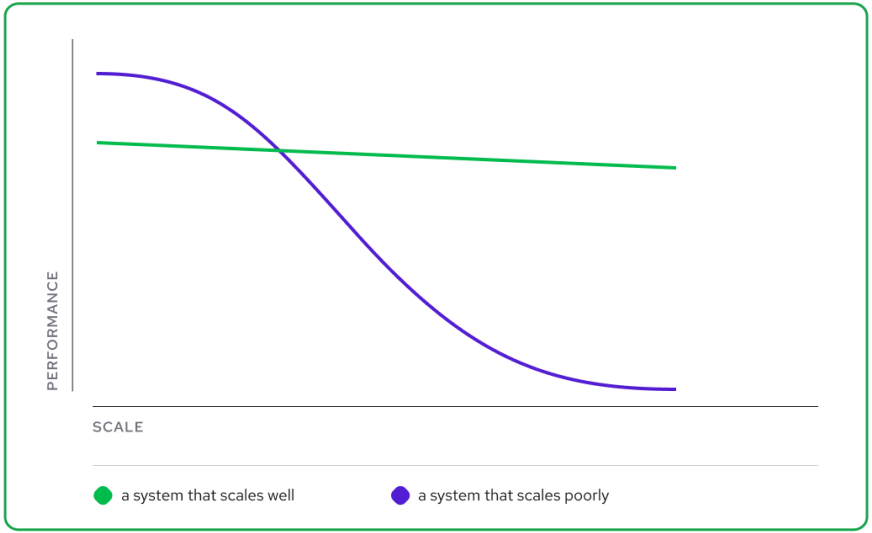
-define terms:
  performance
  scale
  efficiency
- define terms:
  parallelism
  concurrency

-real life scenario:
  bad decisions (accept late requirements)
  bad (re)design
  improve design
- future improvements



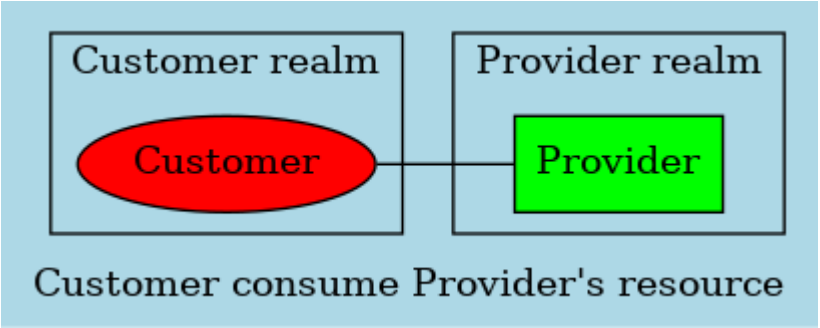
Definitions

- Scale
  - Nr of customers (10^6)
- Performance
  - Throughput
  - Delay & jitter
- Efficiency
  - Control costs



Basic operation

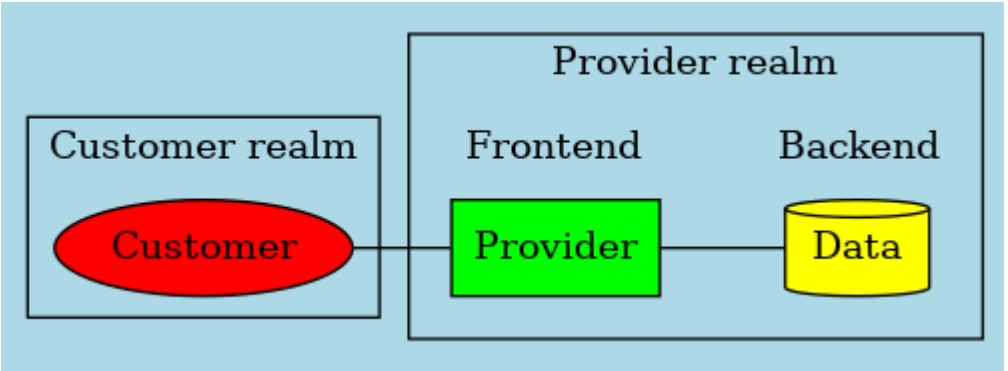
- read or write
- blocking (sync)
- time bound



Basic operation, sync, cache miss

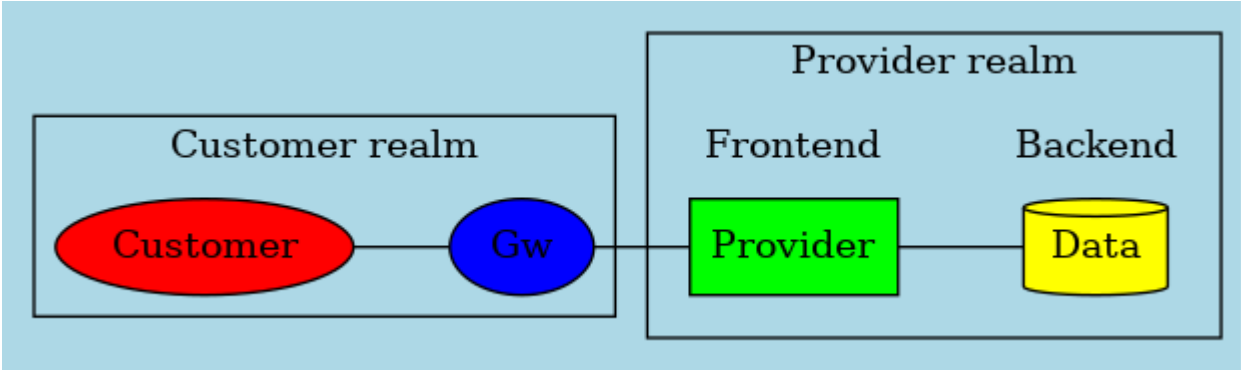
- front end facing customer
- front end hides

  back end
- front end as customer for back end
- timers add



Customer’s GW

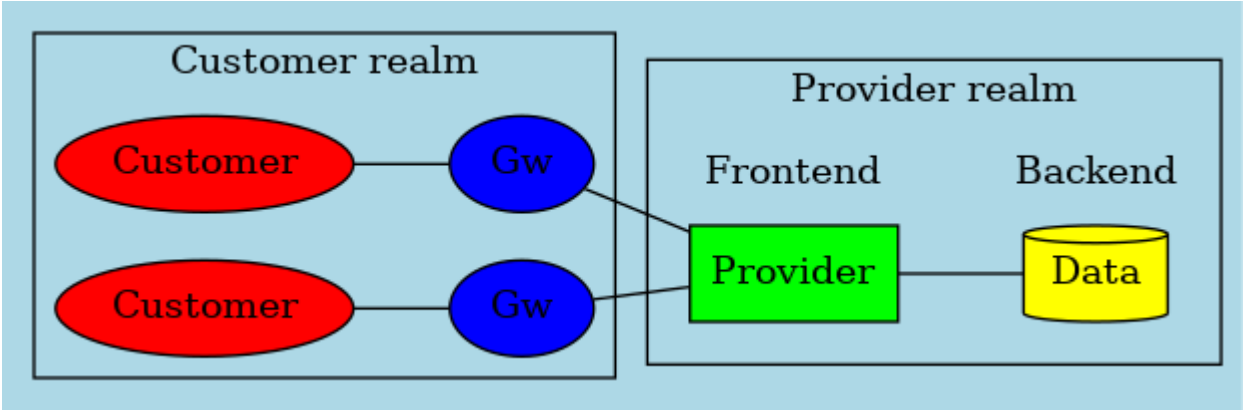
- Gw
  - an appliance
  - an app
- connects customer to the outside world
- generic or provided by service provider
- can give user information about progress through a feedback loop from Provider’s front end



Multiple customers

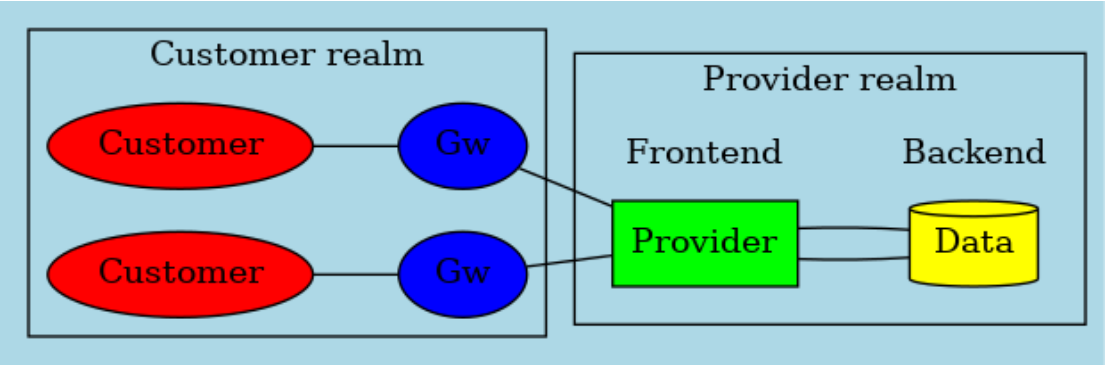
- Simultaneous access attempt
  - Provider’s Front end is the single

    point of access
  - Connections multiplexing
    - space/resources MUX ,aka parallelism
    - Time MUX aka concurrency



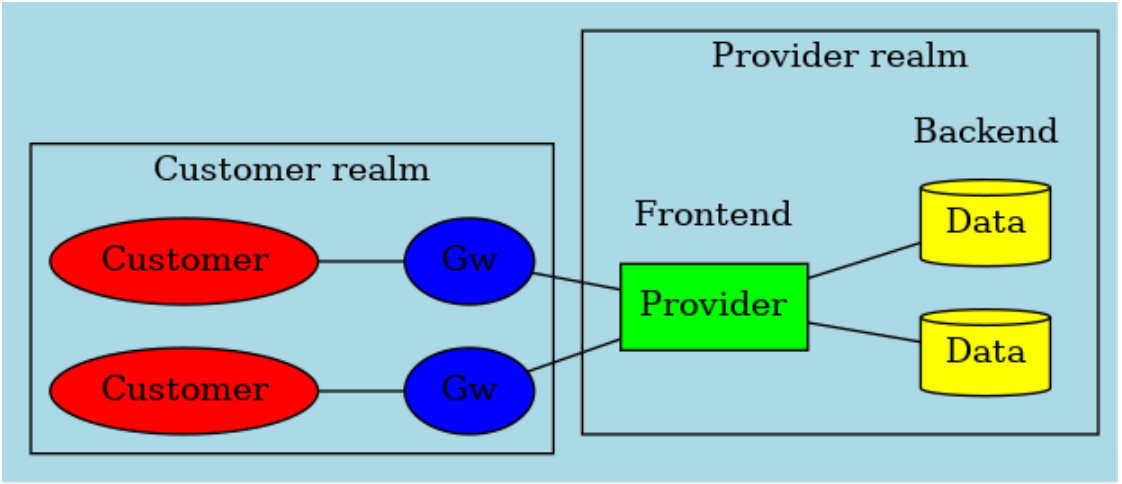
Multiple customers – parallelization

- Simultaneous access attempt
  - Front end spawns a handler for each customer connection
  - Needs dedicated hardware: CPU, memory, BUS



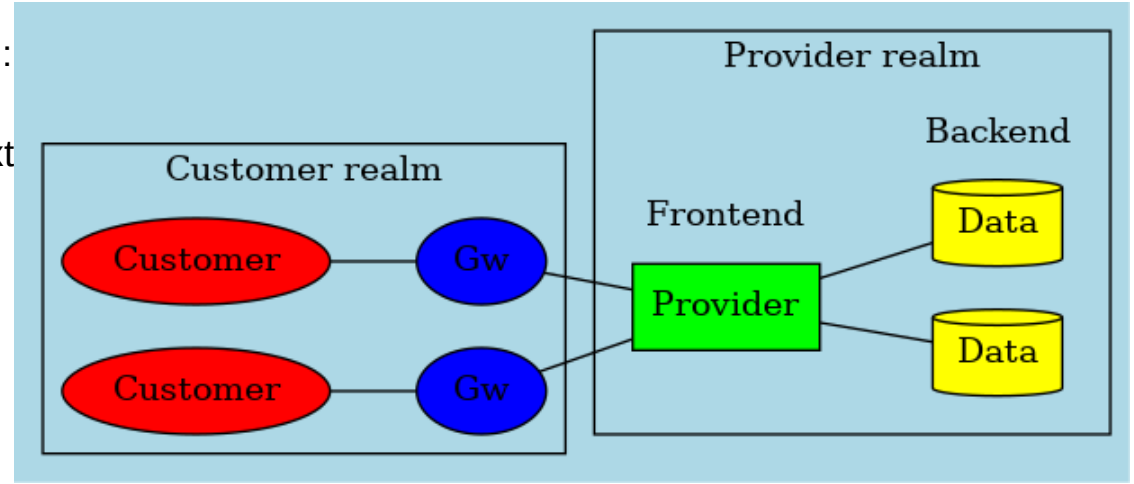
Multiple customers – parallelization

- Code stays simple
  - 1 handler for each customer
  - multiple handlers in parallel each consuming resources (CPU, RAM, BUS, IO)
  - can block
- Single front end multiple back ends
- Front end becomes single point of failure
- Front end (MAY) become the bottleneck
- Front end prone to be I/O bound
- Parallelization does not scale
  (c10k, thundering herd etc)



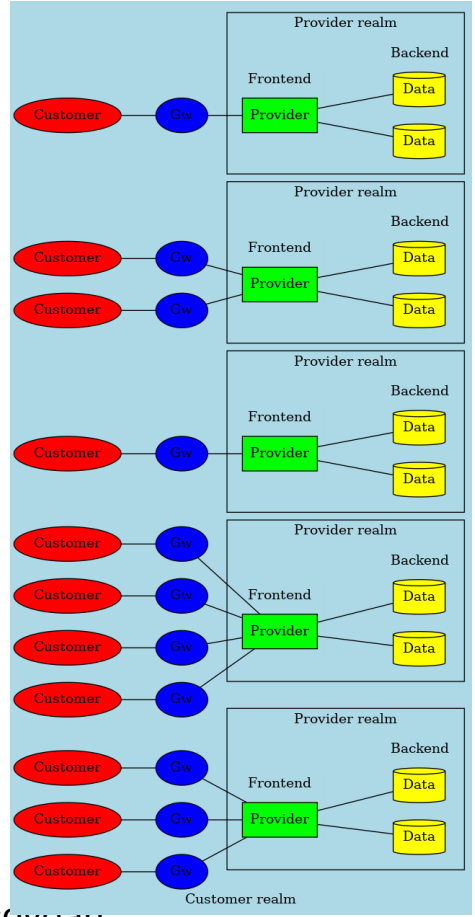
Multiple customers – concurrency

- split time in slices and multiplex
- use idle times caused by slow I/O
- completely different way of communicating:
  - Send message, don’t wait for response
  - Set a callback function and save context
  - event based communication (async)
  - CANNOT BLOCK!!!
- Must get context for each message.
- Store context locally as hash OR carry context with message (uncommon)
- Front end becomes bottleneck because of CPU used for retrieving the context.



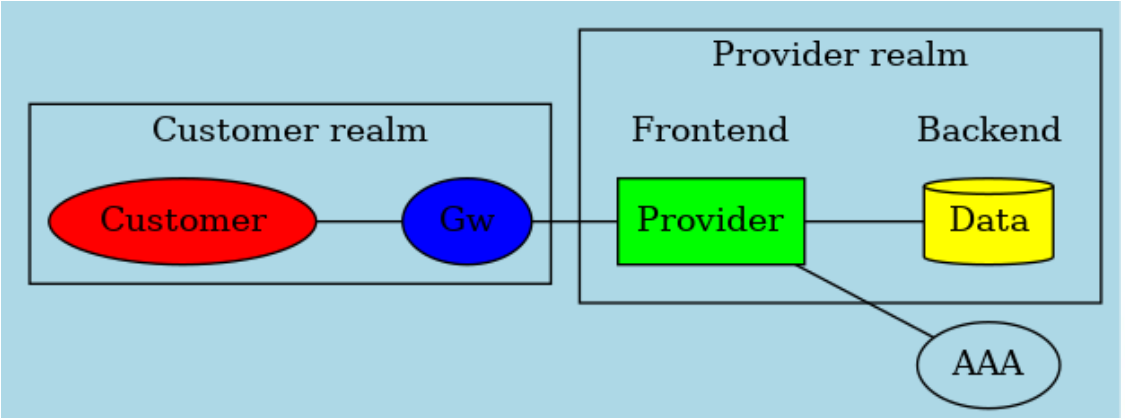
Distribute Front end

- Since Front end became bottleneck – split it. How?
  - DNS round robin
  - Anycast routing
- Opportunity to get closer to the customer (improve latency & jitter)
  - Multiple points o presence across the network graph
  - If one node fails, others can take over transparently
- Distributing brings load sharing AND redundancy
- How good is the hashing? Even distribution?
- Cost effective. Cheap. Scales well.



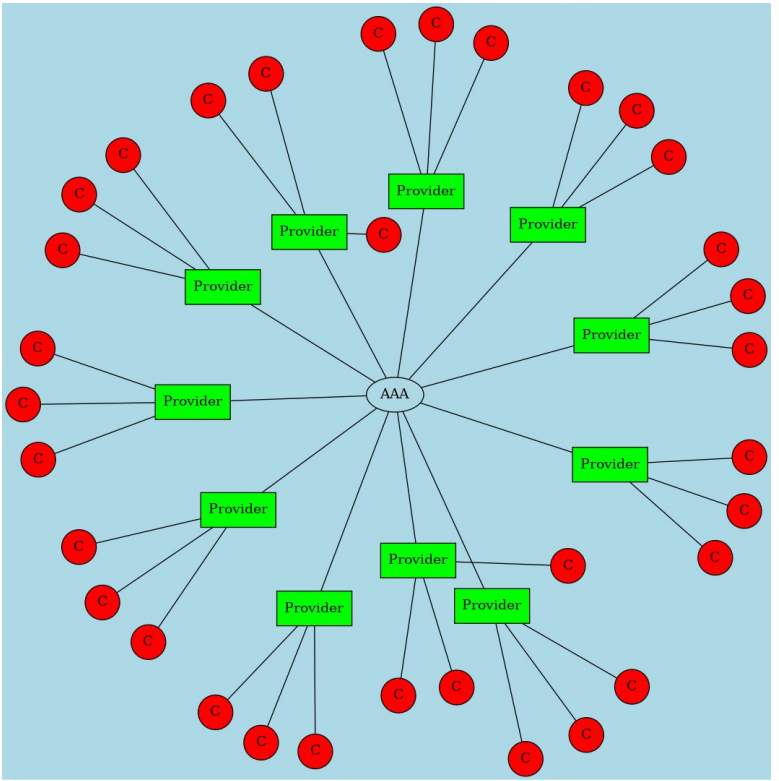
AAA

- Authentication
- Authorization
- Accounting
- Front end communicates with AAA
- AAA impact on perception of quality
- Keeping AAA in sync with reality issue. Scalability.



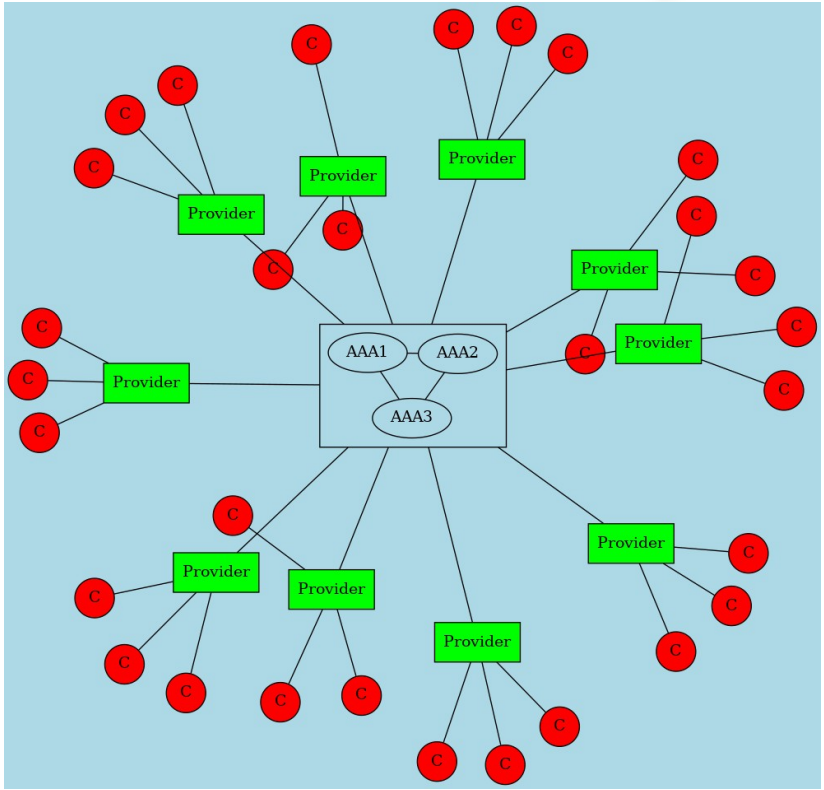
AAA at scale

- 10^6 users
- 10^2 back end
- 10^1 front end
- Dozens of Front ends hammer the AAA simultaneously with hundreds of queries each
- AAA becomes the bottleneck. Scale issue.
- The Accounting is flooding



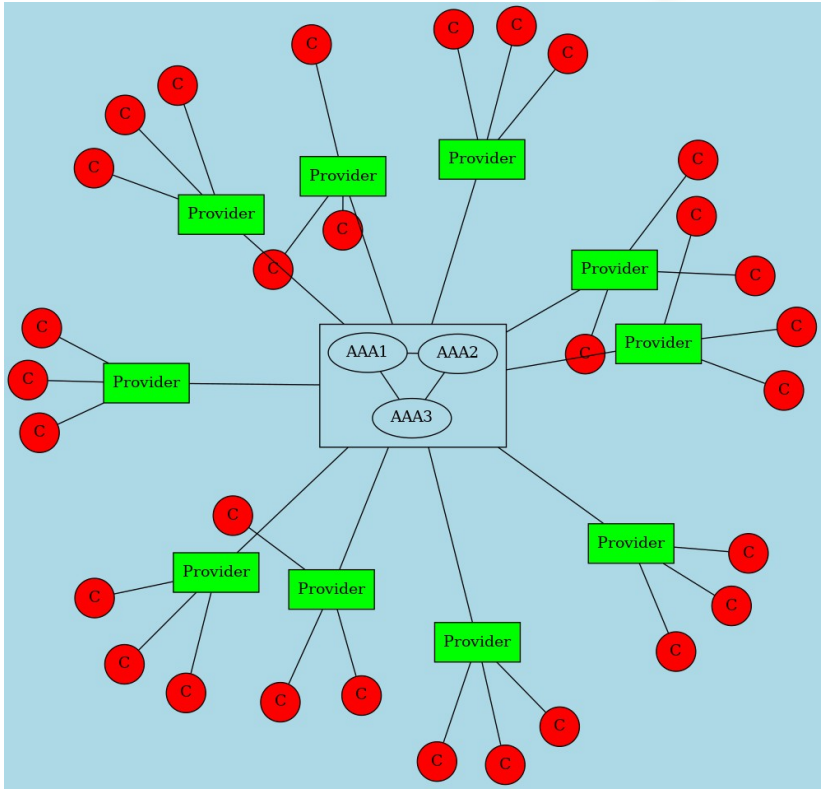
AAA - MongoDB replica set

- BLOCKING!!!
- Replica set nodes, geographically spread (delay)
- Write concern
- Majority
- Solution has good performance
- What happens when large number of customers “disappear”
- thunder created by Accounting impacts Authenticate & Authorize
- Solution does not scale



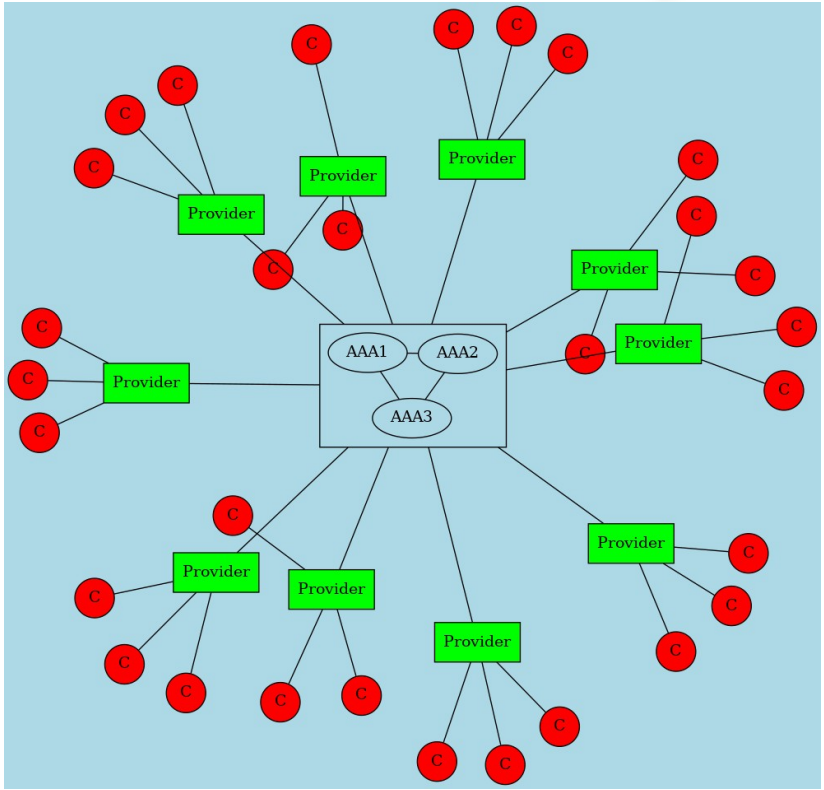
AAA – RabbitMQ to the rescue

- One server instance on each Front line - local
- non blocking
- elastic
- decouple time critical services from time consuming ones
- periodically gather the data in queue and inject it remotely in MongoDB
- opportunity to coalesce queries
- time stamping records in rabbit queues brings ability to reproduce events later



RabbitMQ & MongoDB future improvements

- Aggregate RabbitMQ queues even more on the AAA server before generating MongoDB queries
- Rework MongoDB collections to allow limit-less aggregations



Thank you…

Q&A