Skype's Journey From P2P: It's Not Just About the Services

Bruce Lowekamp
People and Connections
June 27, 2018

Microsoft's Intelligent Conversations and Communications Cloud (IC3)
Powering Skype, Teams, and O365

Skype History

- First released in 2003
- P2P, based on Global Index originally used for KaZaa file sharing
- Chat, audio, video, file sharing, contact invites all over P2P
- Acquired by Microsoft in 2011
- Supernodes moved to datacenters
- Chat moved to (evolution of) Messenger chat service
- Calling, file-transfer, contacts, etc moved to new services
- P2P network officially being decommissioned in Fall 2018

Outline

- Original P2P architecture
- P2P compared to Modern service architecture
- Why not P2P?
- Migrating from old to new architectures
- Doing it well: Experimentation at massive client scale

Skype P2P Architecture

P2P Network formed by clients

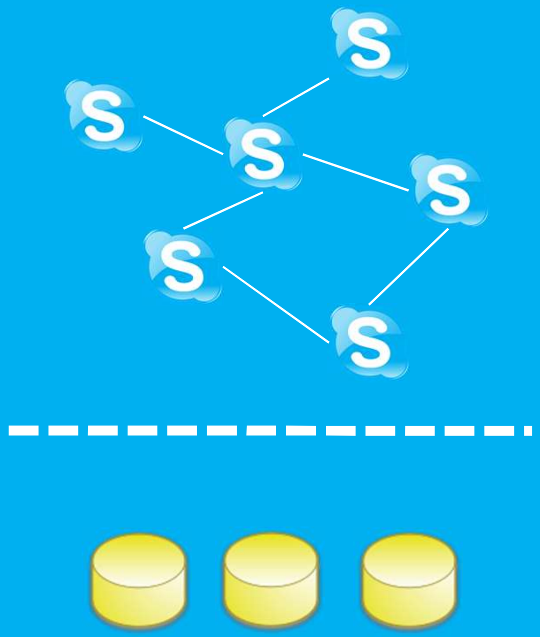
Backend team running mostly DB-based services

Shared Library with clients (data structures, etc)

Services were thin shim on top of sharded PG SQL

PG bouncer: Transparently sharded stored procedures

LUX + DUB



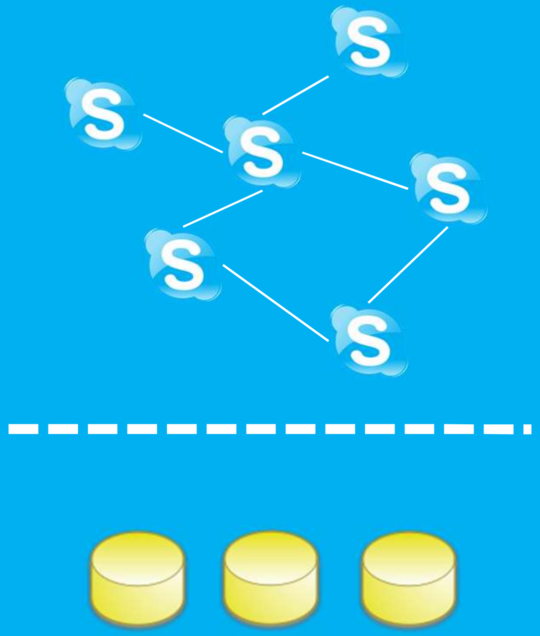
P2P Contact Invites

Search for users across SNs

Send invite (signed) to target via P2P

Receive signed ack with secret.
Update local and feed to other nodes

Lazy sync to backend



High Availability in P2P

P2P Network implements HA

- Invites easily sent when both clients online

Backend forwards P2P invite

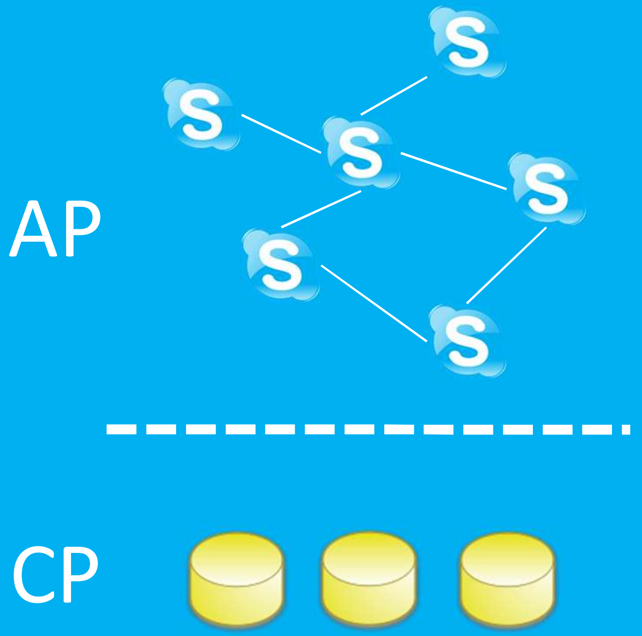
- When invitee offline

Operation completed by clients

Changes to contact list lazily synced to DB

CAP Theorem

- P2P Network is AP
- BE DBs are CP



Breaking apart P2P Contact Changes

"Changes lazily synced to DB"

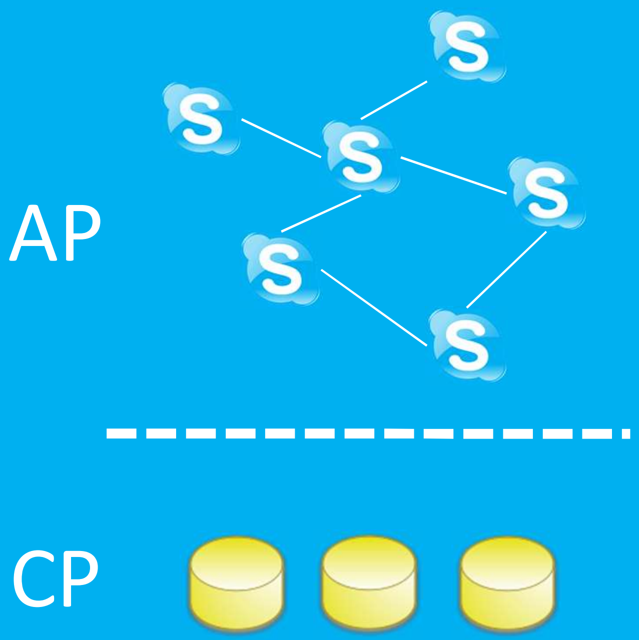
Sequence of changes sent to clients and DB

DB syncs to clients

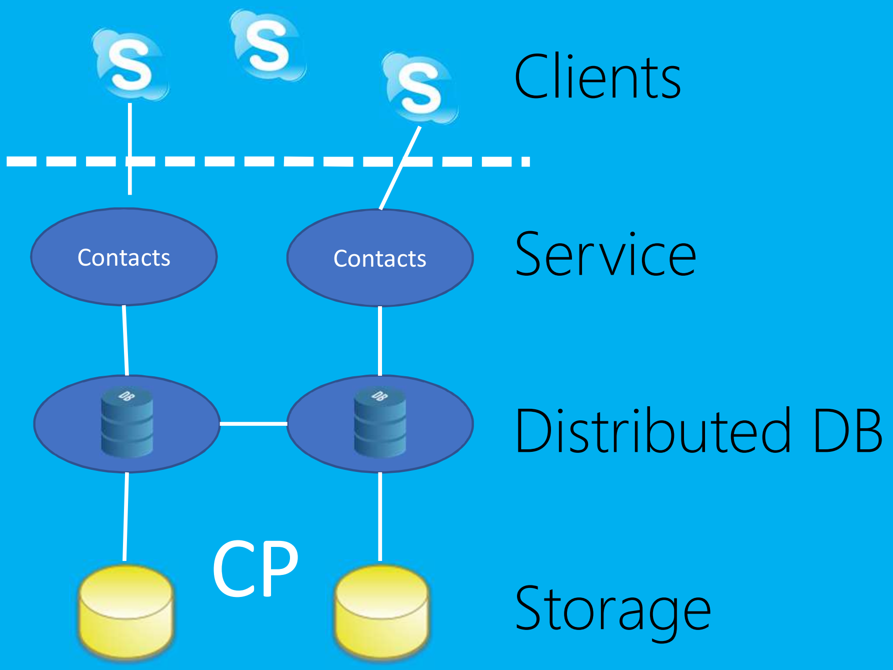
Eventually all DB and clients see same result

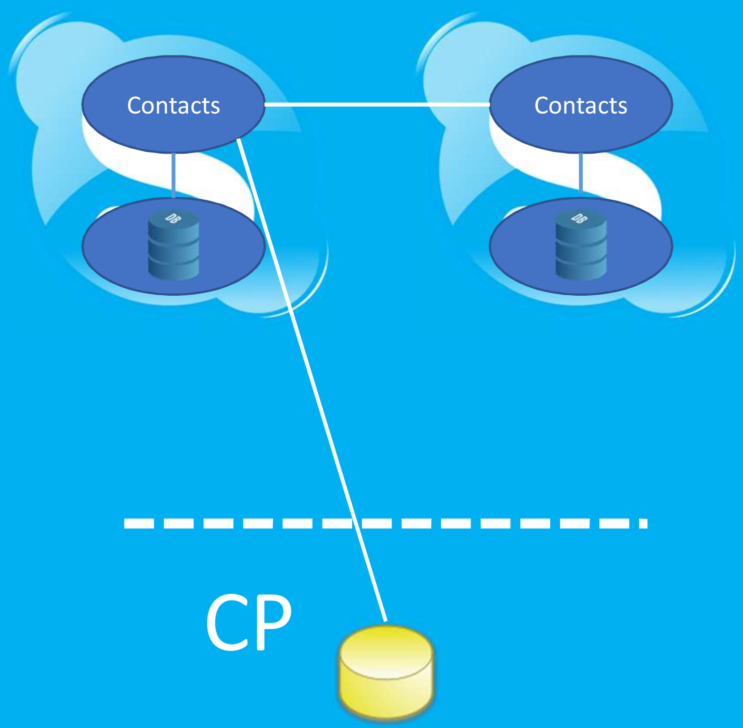
CRDT?

"J" in JCS was for Journaled



Distributed Service vs P2P Architecture





Why not P2P?

Desktop apps no longer dominant
Servers cheap
Need to support mobile

Offline messaging, suggestions, server-side search, browser state

Business logic (and service implementation) in clients, not services

Can still do P2P media and E2E encryption in service-based systems

Migrations

Supernodes->Dedicated Supernodes->Trouter

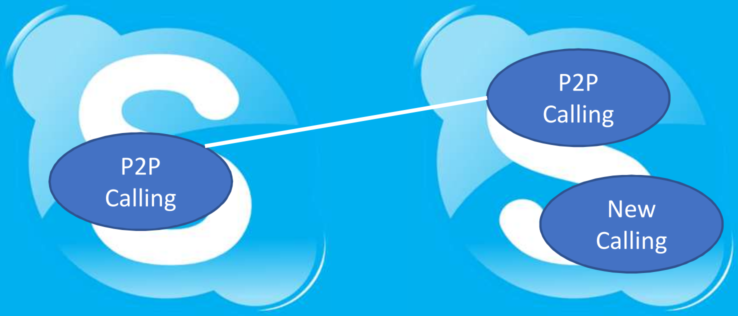
Chat: P2P -> P2P+Griffin -> Messenger -> New Chat Service

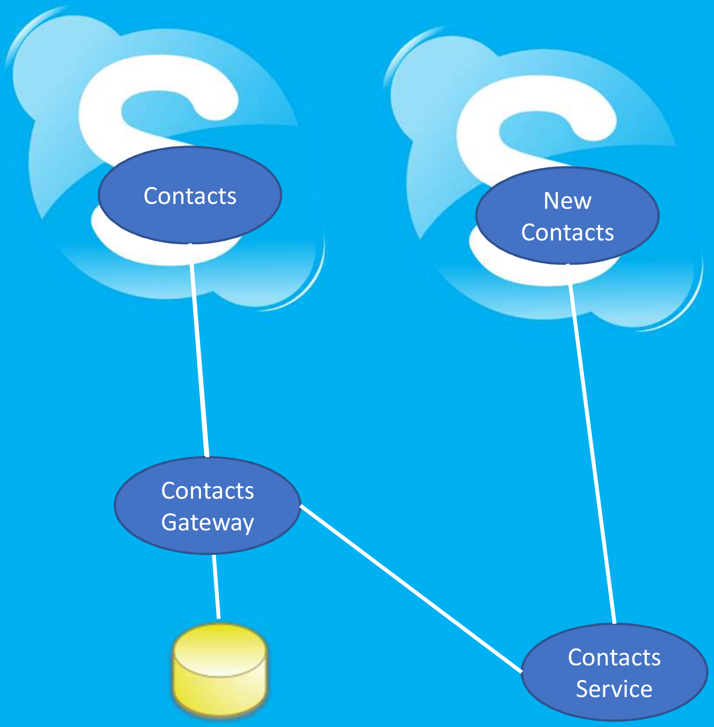
Contacts: CBL->JCS->ABCH->PCS->EXO

Calling: P2P -> NGC

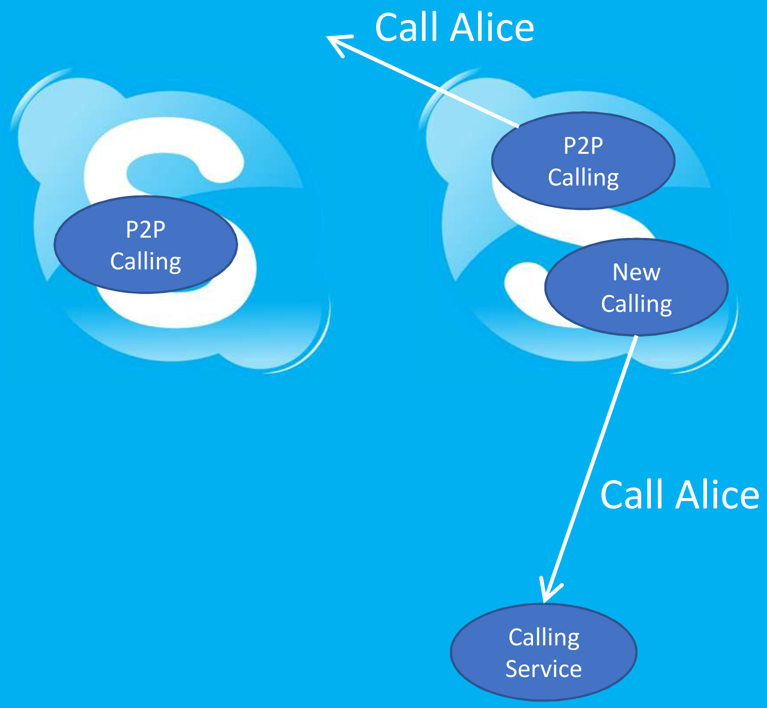
Login: Skype -> MSA

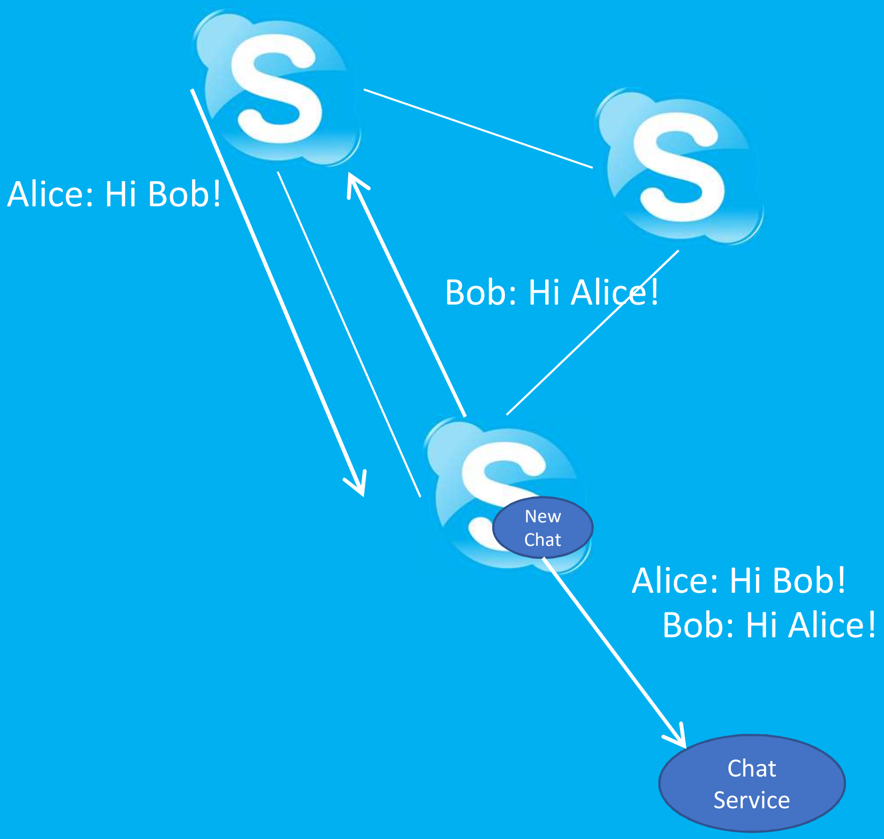
Dual-head vs Gateway





Dual-Stack: Calling and Chat

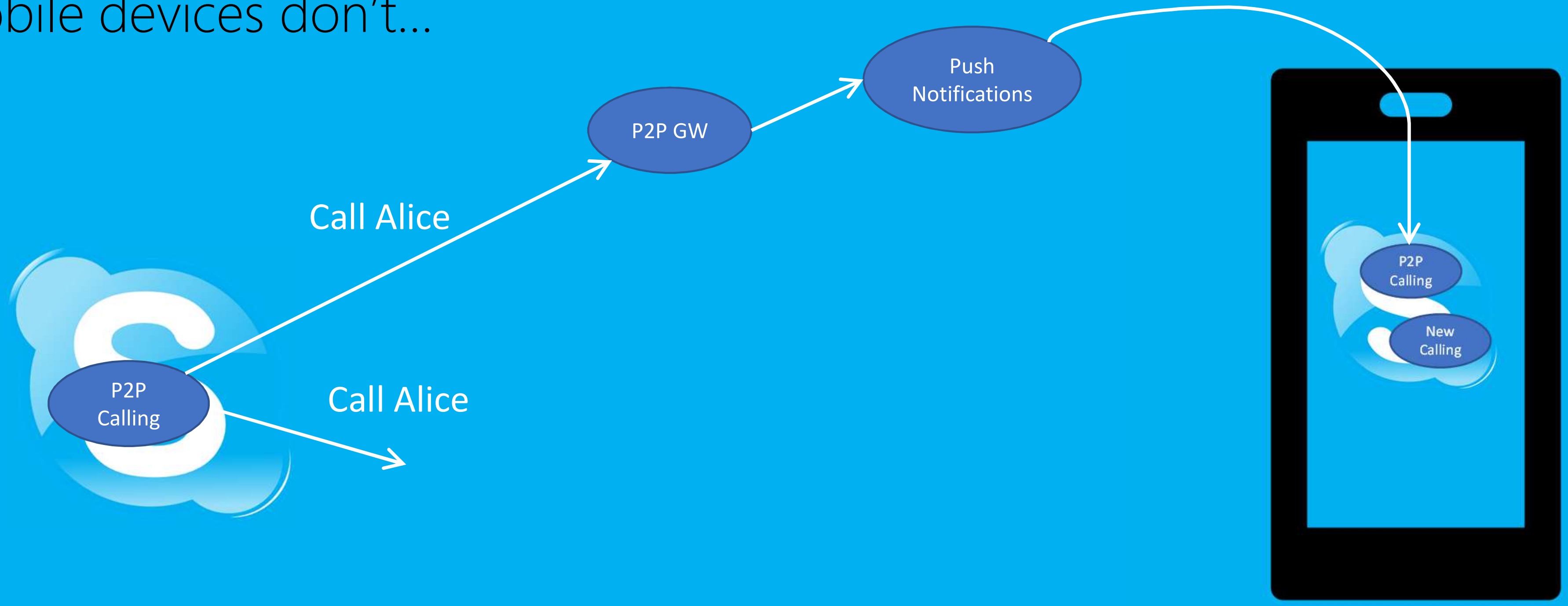




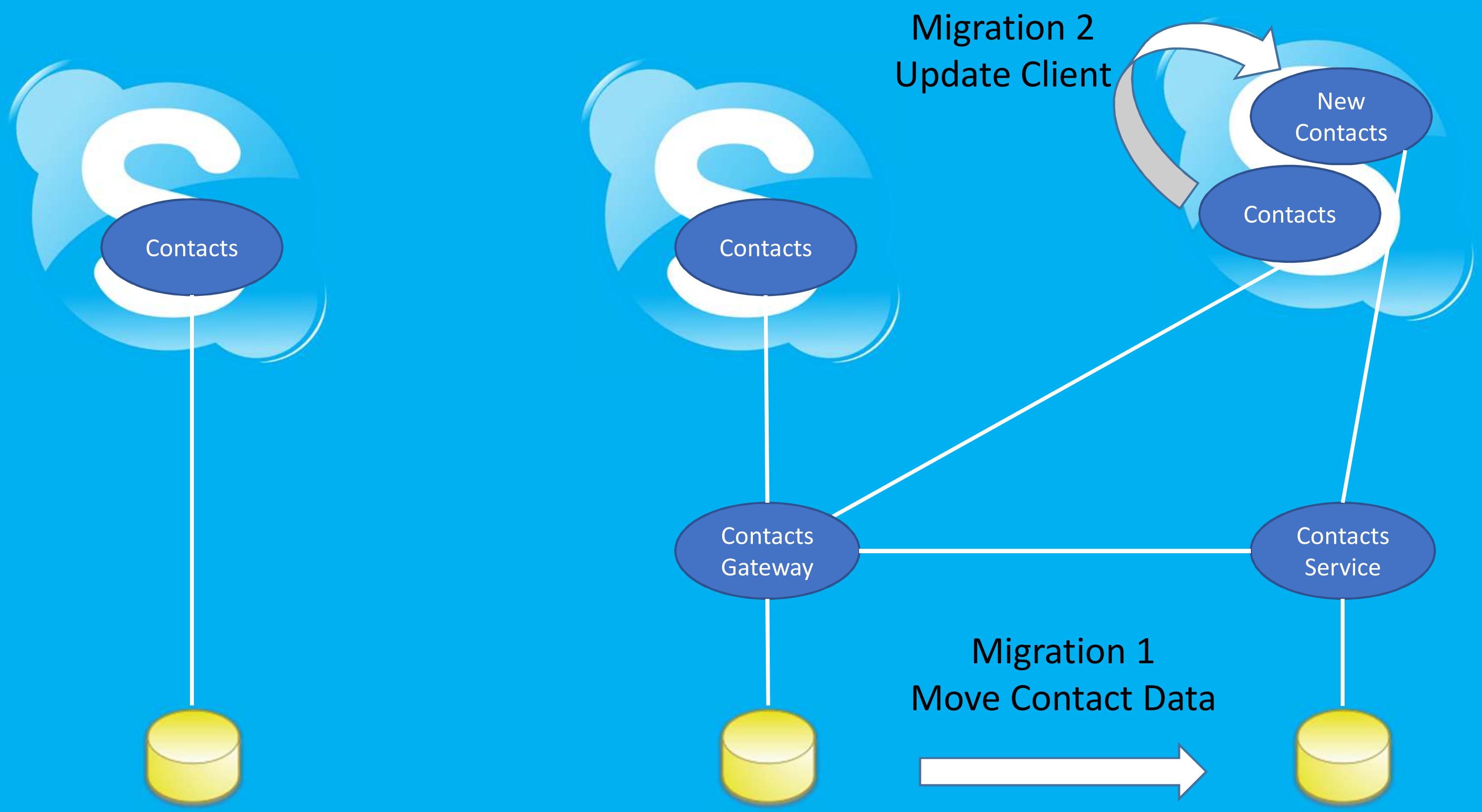
Technology gateway: Dual-headed with Help

P2P requires clients running continuously

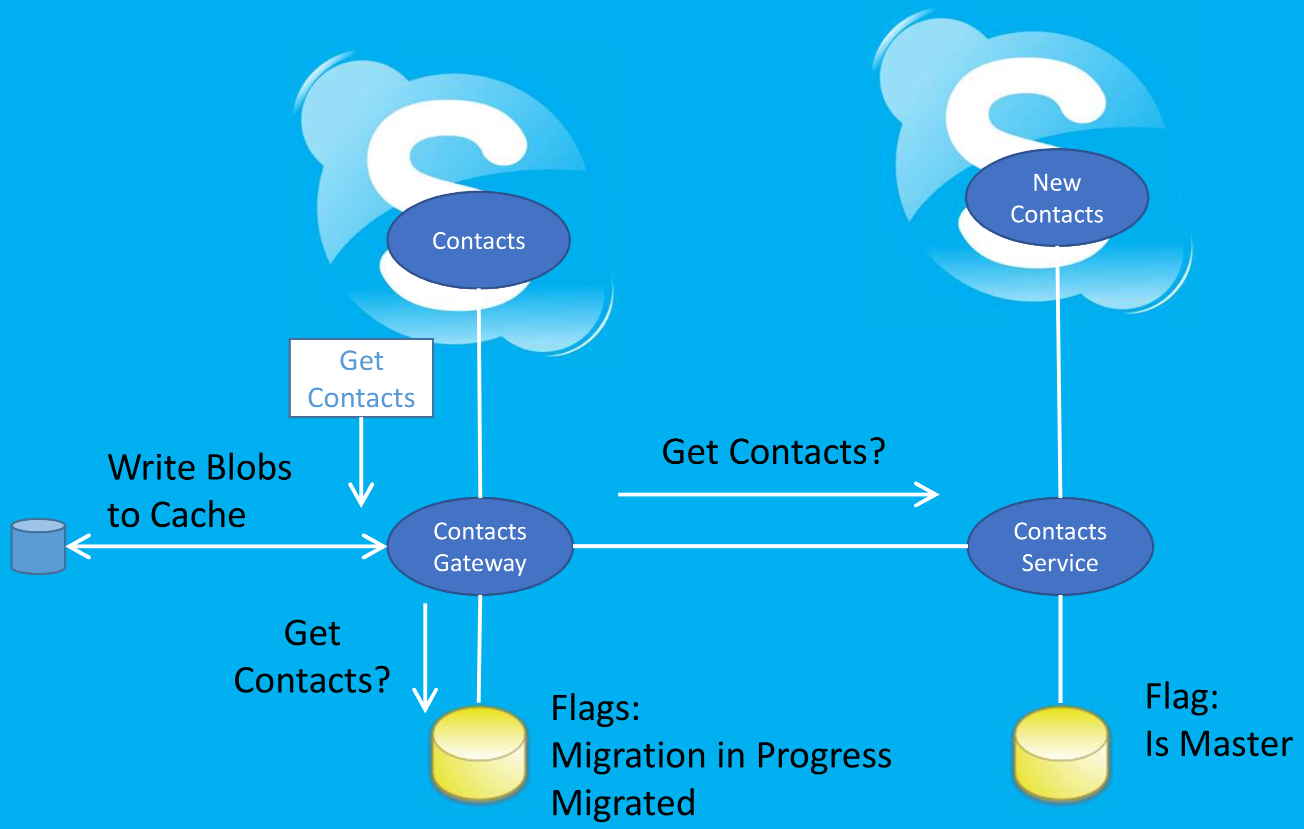
Mobile devices don't…



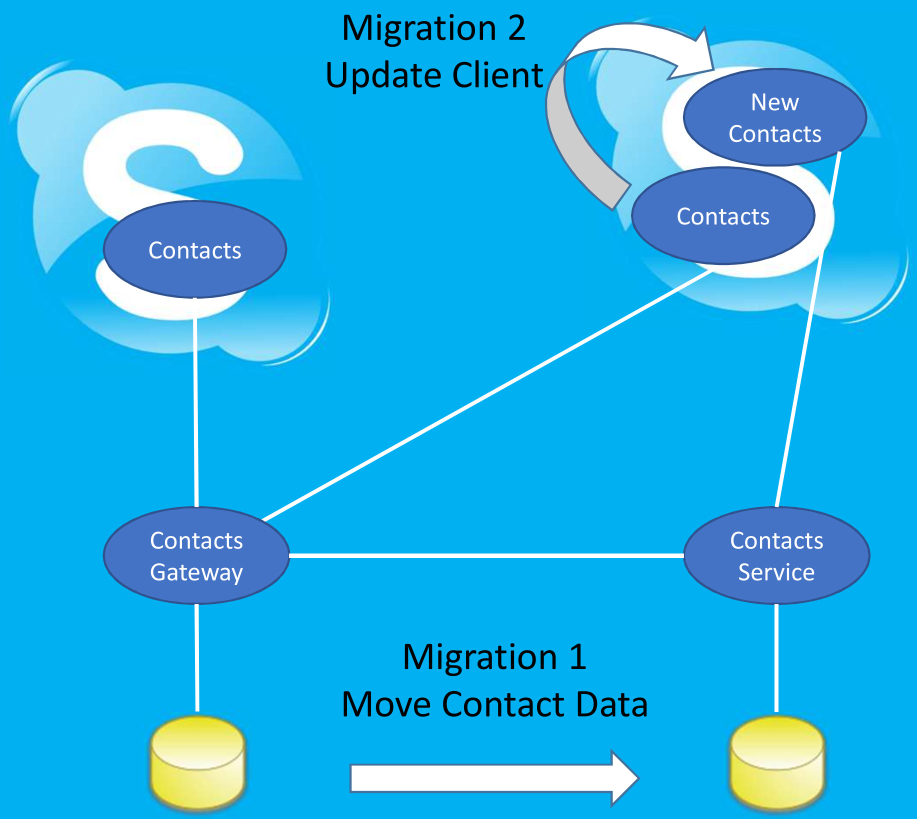
Gateway for Contact migrations



Contact migrations



When to migrate?



Need for Online Experimentation

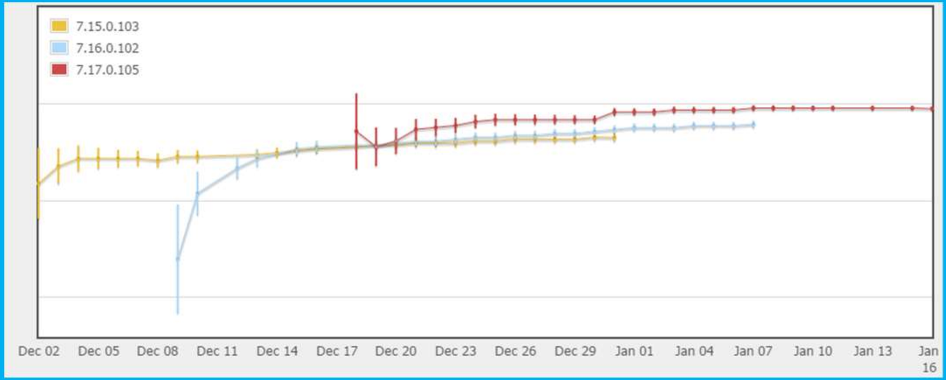
Even objective metrics are a function of

- Product quality
- Seasonal/weekly effects
- User population
- Device population
- Usage scenario

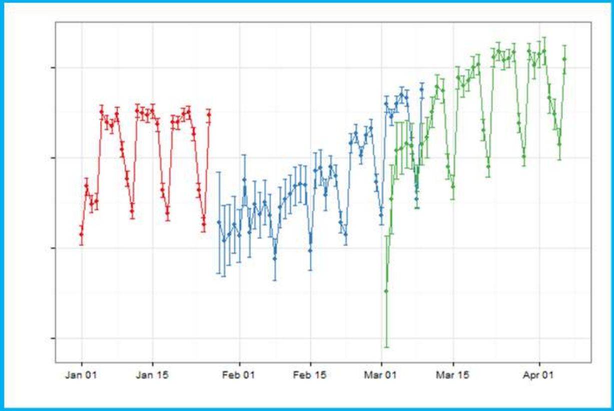
These aren't stable across new client releases

Need robust online experimentation to separate new calling implementation from other factors.

Early Adopter Bias



Seasonality, Overall Trends



Experimentation – When to use A/B Testing?

When to use A/B testing:

- Making a data-driven decision about the impact of a change in the product

How is A/B testing different from "monitoring metrics before and after a change":

- A/B testing is the only valid method to draw causal inference – i.e. the changes in metric behavior cannot be attributed to any particular change in code unless in a randomized treatment assignment (A/B testing) setting

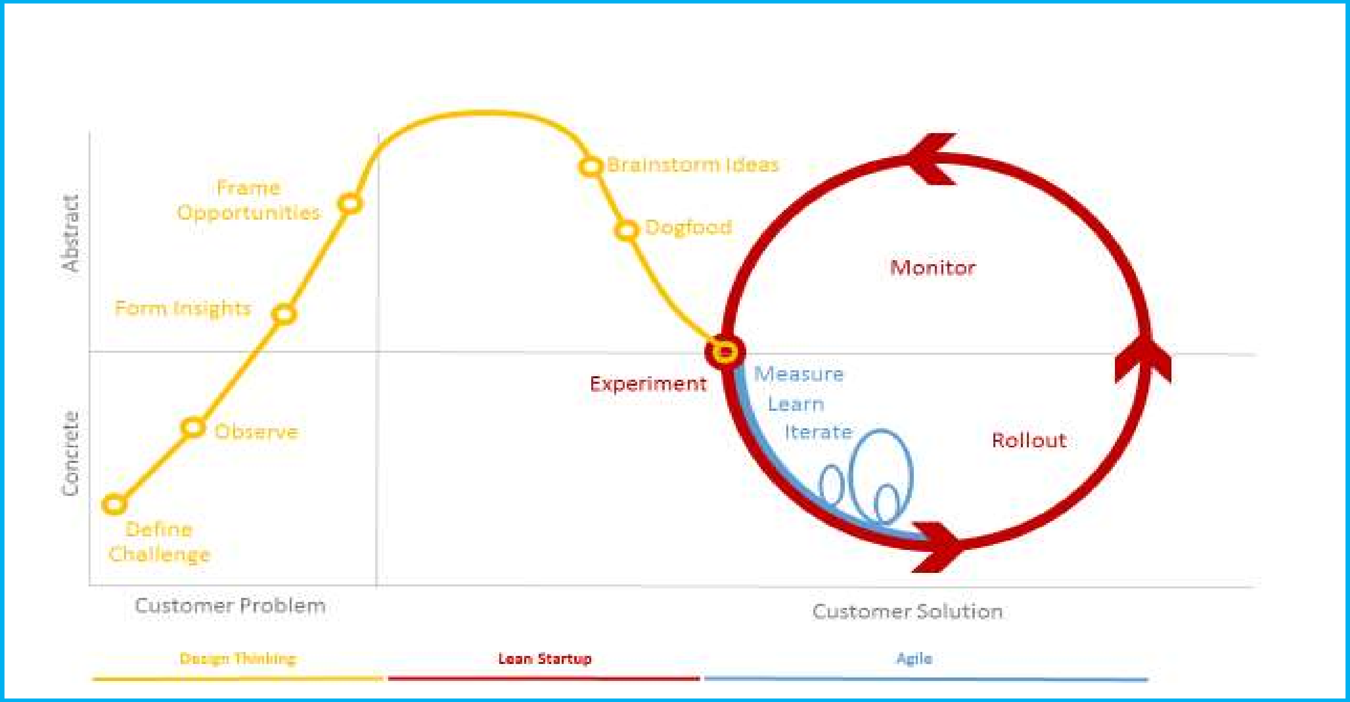
Why set up automated scorecards vs manually aggregating data into test statistics:

- To make sure the results are trustworthy – it is easy to be misled by data!
- To scale the experimentation so you don't need a data scientist for every single experiment analysis
- To have a standard procedure that controls the rates of false positive/negative in long run over the entire org

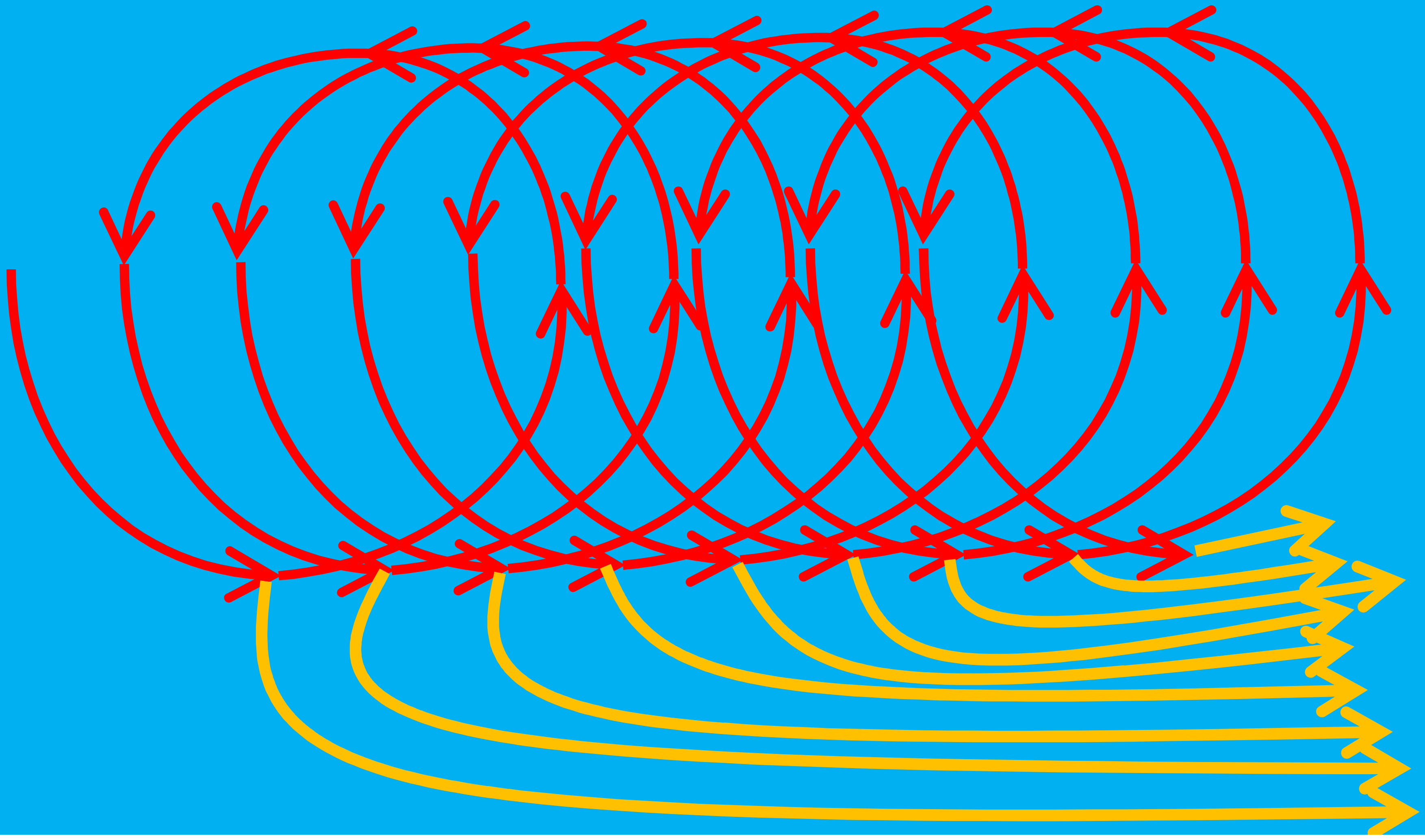
First step for getting started on experimentation:

- Data!
  - Decide about which metrics are to be used for tracking the improvements - they should be aligned with T0 KPIs of the org
  - Make the data available for querying with experimentation labels (e.g. knowing which each calls fell into)
- Link data to a validated scorecard

Experimentation Lifecycle



Experimentation Lifecycle, Client Edition



Experimentation Requirements

Many teams

- Self-service
- Structured Config

Configuration-centric

- Long-lived clients know what, not why

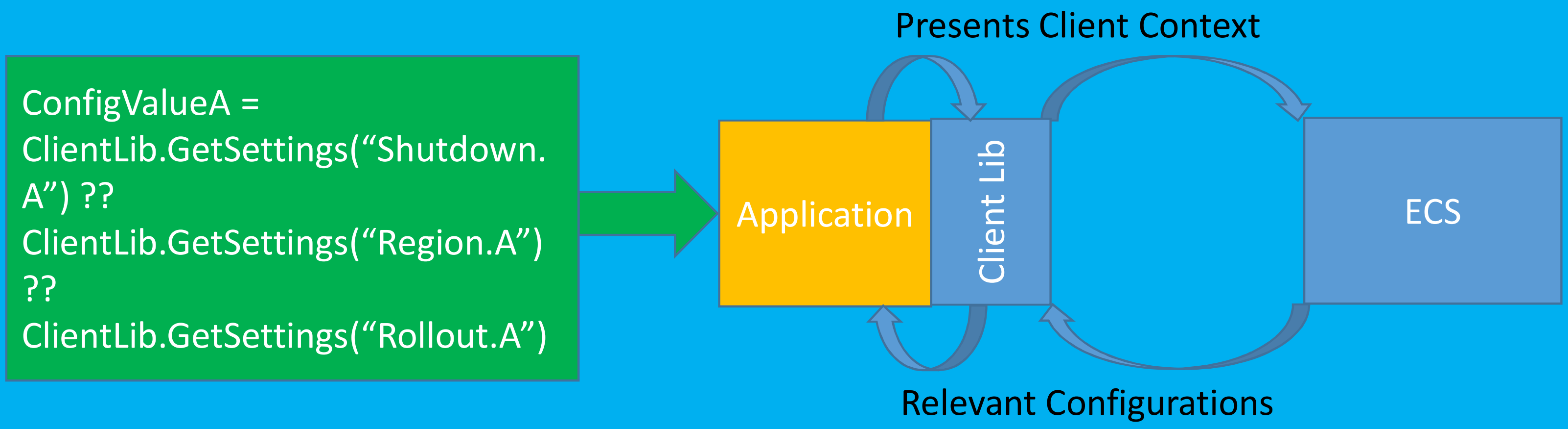
High-quality scorecards

- A&E Experimentation team evolved out of Bing

Experimentation and Configuration Service (ECS) was built to address the flighting and configuration portion of experimentation.

Configuration-Centric View

Straightforward approach gives the client configuration describing its situation, and client decides what to do.



```
ConfigValueA =
ClientLib.GetSettings("Shutdown.
A") ??
ClientLib.GetSettings("Region.A")
??
ClientLib.GetSettings("Rollout.A")
```

Configuration-Centric View

But reasons to change behavior interact

Resolving these collision manually and statically is not scalable

IF Shutdown THEN A=0

IF Country=Australia THEN A=4

IF Ver>2.0 && 80% THEN A=5

IF Version>1.0 THEN A=3

IF Shutdown THEN A=0

IF NOT Shutdown AND Country=Australia THEN A=4

IF NOT Shutdown AND Country != Australia AND Version>2.0 && 80% THEN A=5

IF NOT Shutdown AND Country != Australia AND !(Version > 2.0 && 80%) AND Version>1.0 THEN A=3

Configuration-Centric View

…becomes a Live-site issue

What if the Australia setup needs to be turned off? It is more manageable to disable the precise setup

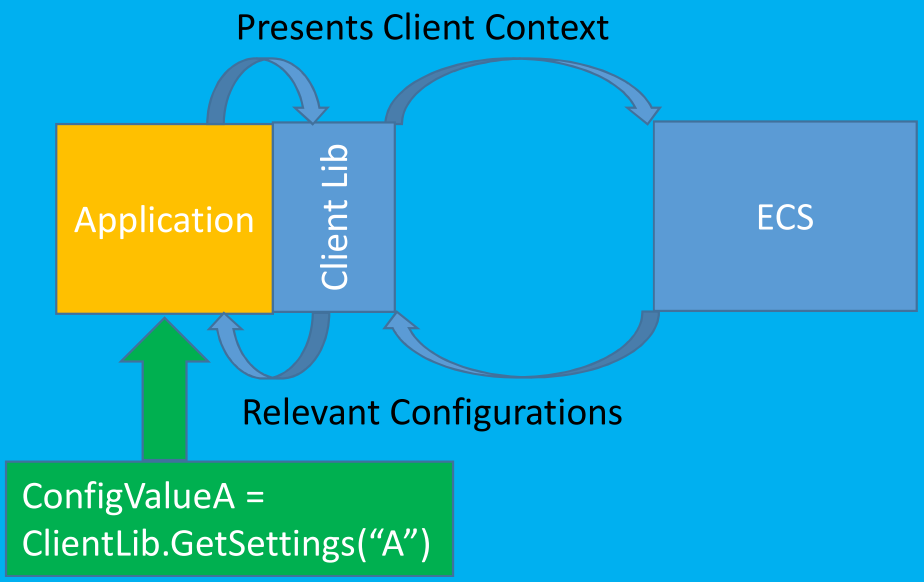
```
IF Shutdown THEN A=0
IF Country=Australia THEN A=4
IF Ver>2.0 && 80% THEN A=5
IF Version>1.0 THEN A=3
```

```
IF Shutdown THEN A=0
IF NOT Shutdown AND Country=Australia THEN A=4
IF NOT Shutdown AND Country != Australia AND
Version>2.0 && 80% THEN A=5
IF NOT Shutdown AND Country != Australia AND !(Version
> 2.0 && 80%) AND Version>1.0 THEN A=3
```

Configuration-Centric View

Applications are made to be Configurable

Applications should only be concerned on What it should be configured to, not Why



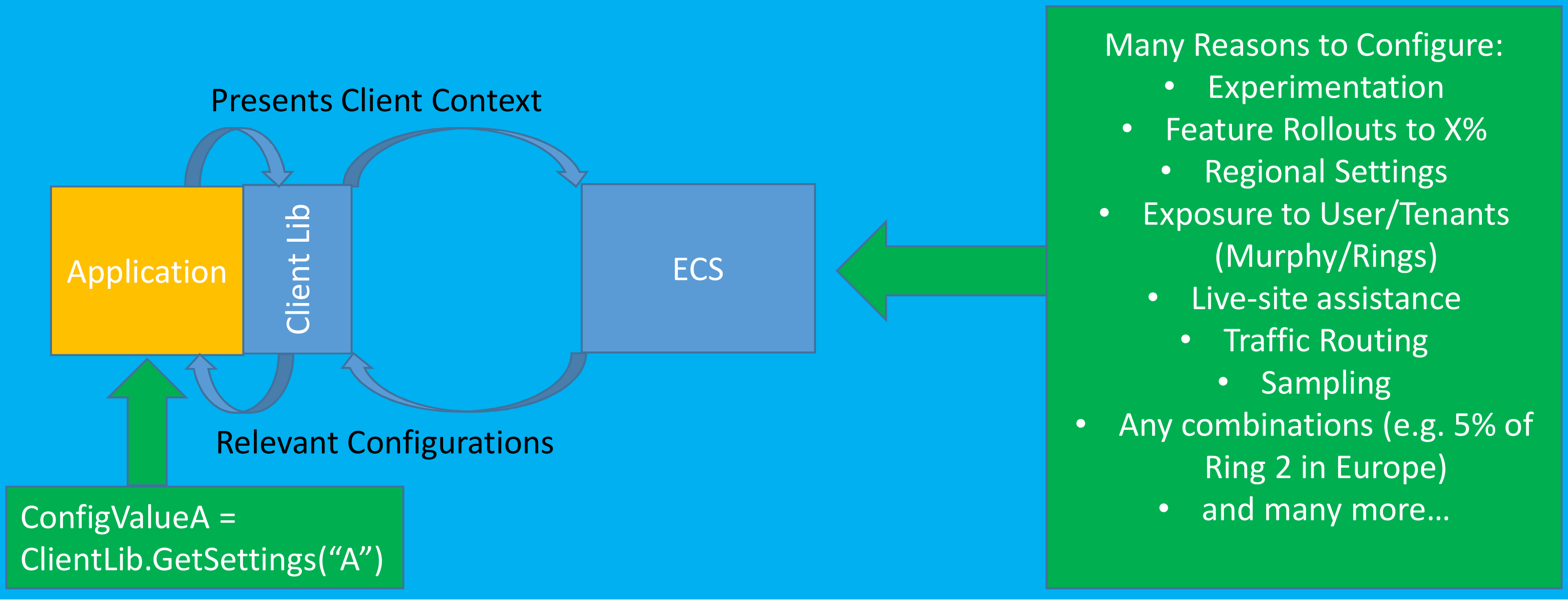
```
ConfigValueA =
ClientLib.GetSettings("A")
```

Configuration-Centric View

And the reason to configure will be many

As the number of reasons scale, the reasons will collide

Need Tie-breaking Rules



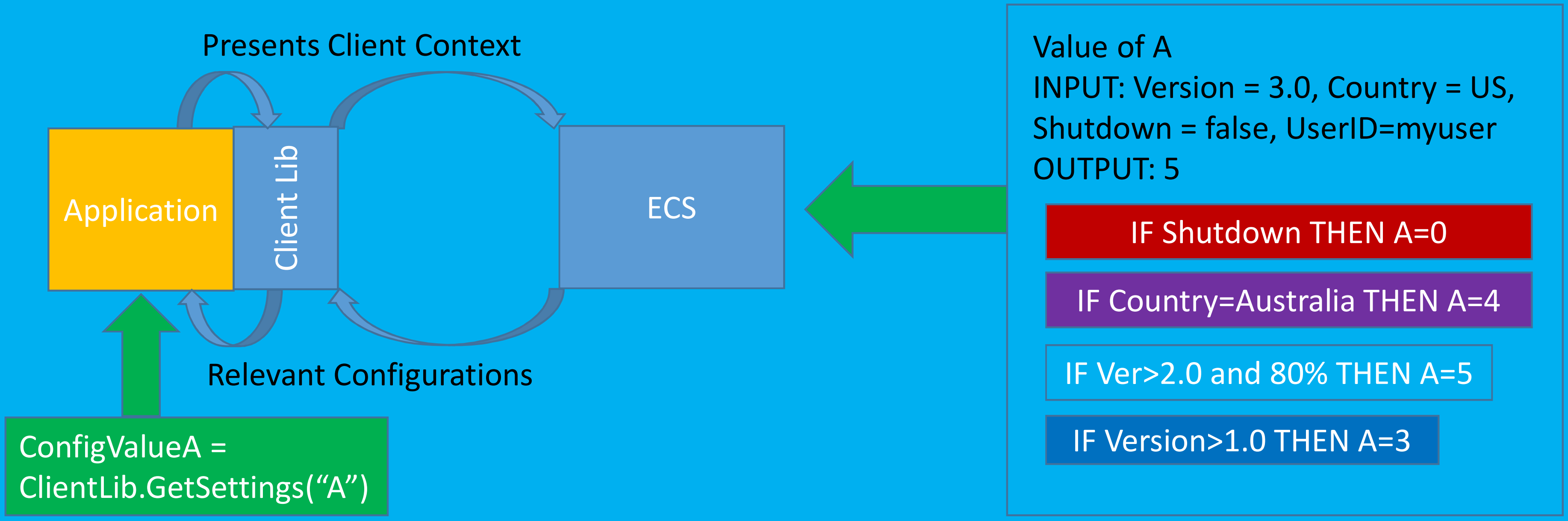
```
ConfigValueA =
ClientLib.GetSettings("A")
```

Many Reasons to Configure:

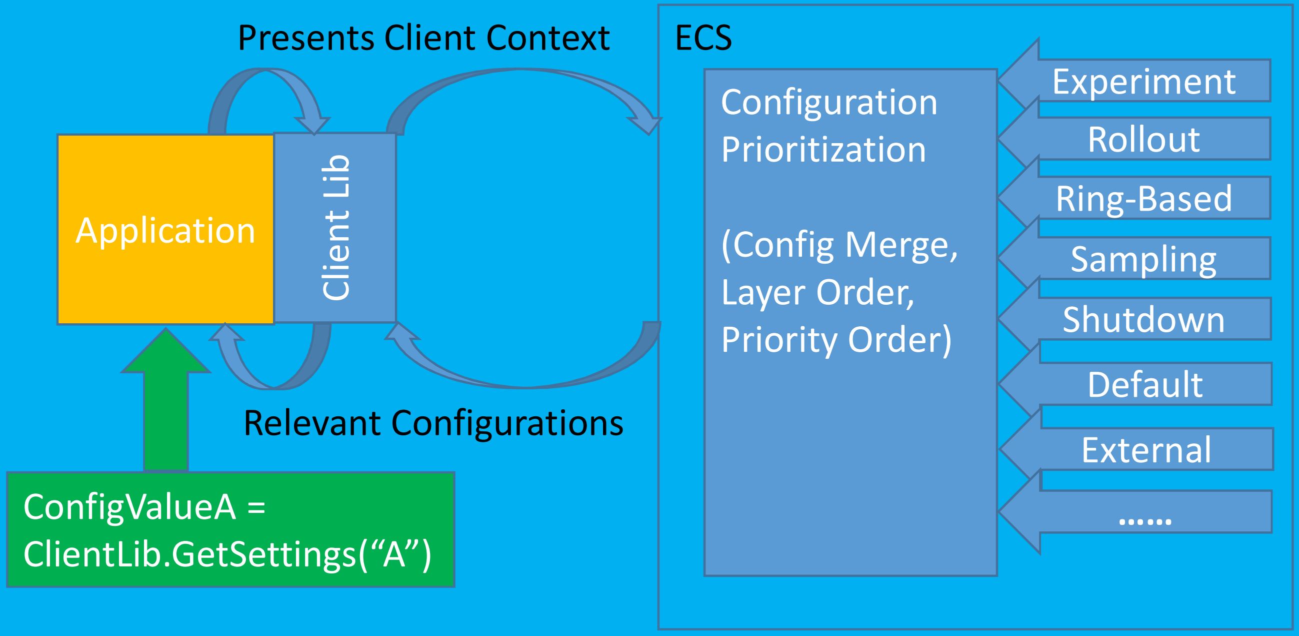
- Experimentation
- Feature Rollouts to X%
- Regional Settings
- Exposure to User/Tenants (Murphy/Rings)
- Live-site assistance
- Traffic Routing
- Sampling
- Any combinations (e.g. 5% of Ring 2 in Europe)
- and many more…

Configuration-Centric View

ECS configuration approach is to provide a set of Tie-breaking rules for users, but let the service resolve the collision dynamically


Presents Client Context
Application
Client Lib
ECS
Relevant Configurations
ConfigValueA = ClientLib.GetSettings("A")
Value of A
INPUT: Version = 3.0, Country = US, Shutdown = false, UserID=myuser
OUTPUT: 5
IF Shutdown THEN A=0
IF Country=Australia THEN A=4
IF Ver>2.0 and 80% THEN A=5
IF Version>1.0 THEN A=3

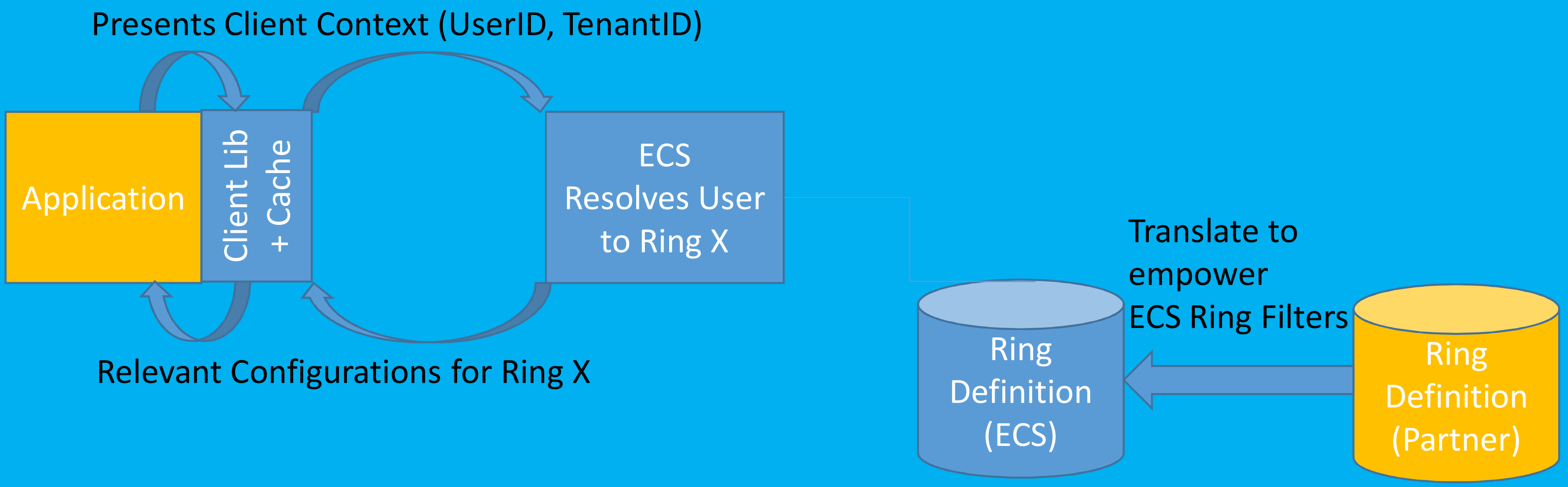
Configuration-Centric View



No Client-Service Contract Change

Example: Configuration with Rings

- Decoupling who the user is from how the application is configured
- No Client-Server contract change. No Mobile re-deployment for Rings


Presents Client Context (UserID, TenantID)
Application
Client Lib + Cache
ECS Resolves User to Ring X
Relevant Configurations for Ring X
Translate to empower ECS Ring Filters
Ring Definition (ECS)
Ring Definition (Partner)

ConfigID

Identify each experiment, rollout, default

Needed for debugging and analysis

<Type-ExpID-TreatID-Iteration>

Configuration Merge

Experiment Config

```
“SkypeAndroid": {
        "ShortCircuit": true
}
```

Rollout Config

```
“SkypeAndroid": {
        "PhoneVerification": false,
        “ShortCircuit”: false
}
```

Merged Config

```
“SkypeAndroid": {
        "ShortCircuit": true,
        "PhoneVerification": false
}
```

ETag

ETag is a hash of the set of ConfigIDs being served

ETag-ConfigID mapping is forwarded to data pipeline by ECS service

Client Telemetry is logged with the ETag

Data Analysis to associate telemetry with an iteration of the treatment

- Client Telemetry.Etag
- Data Analysis.ConfigID
- Service log: Etag-ConfigID Map

Also useful for debugging client implementation

Impression-based vs Sticky

Conventional experimentation:

- Numberline assigns user to experiment. Experiment is sticky.
- Analyze impact over time

What if your experiment is more risky?

- Next-gen code frequently known not to be better (yet)
- Still need to get real-world experiment
- “Impression-based” assign at random each fetch
- No one gets broken experience for more than an hour/restart

Importance of Scorecards

Changes in important metrics

ALL metrics, not just intended by experiment

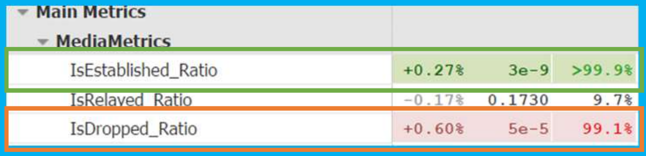
P-values of changes to confirm caused by experiment

Unanswered call UX experiment

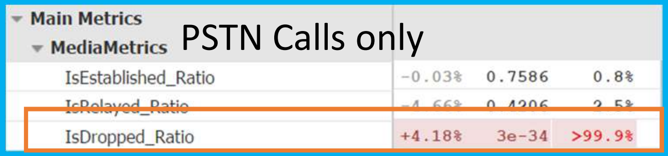
- Higher ratio of established calls
- BUT, Call Drop Ratio is up by 0.07% overall, caused by PSTN drops

Likely explanation: retrying a failed call on PSTN isn't useful on a bad network

Experiments can have unexpected consequences on other scenarios. A scorecard capturing important metrics across all scenarios is needed to find unintended consequences.



Main Metrics			
MediaMetrics			
IsEstablished_Ratio	+0.27%	3e-9	>99.9%
IsRelayed_Ratio	-0.17%	0.1730	9.7%
IsDropped_Ratio	+0.60%	5e-5	99.1%



PSTN Calls only

Main Metrics			
MediaMetrics			
IsEstablished_Ratio	-0.03%	0.7586	0.8%
IsRelayed_Ratio	-4.66%	0.4206	2.5%
IsDropped_Ratio	+4.18%	3e-34	>99.9%

ECS Today

Scale (as of 6/8/18)

479 Project Teams

Currently running:

- Experiments 388
- Rollouts 2.74K
- Defaults 701

12.69K Complex Configs

3.83K layers (uniquely salted numberline)

~140K RPS (daily peak)

Used by Skype & Teams clients and services. Most Office apps, etc...

Lessons from Skype’s evolution

P2P

- Architecture is different, but same HA principles can be achieved
- Solved problems originally, but became a bottleneck over time

Migrations

- Config support plans for your next migration in advance
- Pick strategy based on complexity of transition

Experimentation

- Migration (and other changes) require robust experimentation
- Don’t bake in experiments: What not Why!

Acknowledgements

Many, many people at Skype and Microsoft built the systems described here and implemented the strategies to migrate users to newer systems.

Special thanks to Eric Lau, Michael Rubin, Daniel Schneider, and the ECS team. The E2E experimentation pipeline includes major components developed by the Aria, A&E EXP, IC3 Media, and other partner teams.

bruce.lowekamp@skype.net

https://linkedin.com/in/brucelowekamp