iRobot

# Challenges for Serverless Native Cloud Applications

## Third International Workshop on Serverless Computing

**Ben Kehoe**

Cloud Robotics Research Scientist
AWS Community Hero
@ben11kehoe

2018-07-02

# About me

- Cloud robotics: Connecting robots to the internet to help them do more and better things
- Cloud Robotics Research Scientist at iRobot (since Feb 2015)
- PhD in Mechanical Engineering from UC Berkeley, Dec 2014
  - “Brass”



# 2015


iRobot
NASDAQ
stem
1990
1991
1996
2001
2002
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015

Roomba® 690

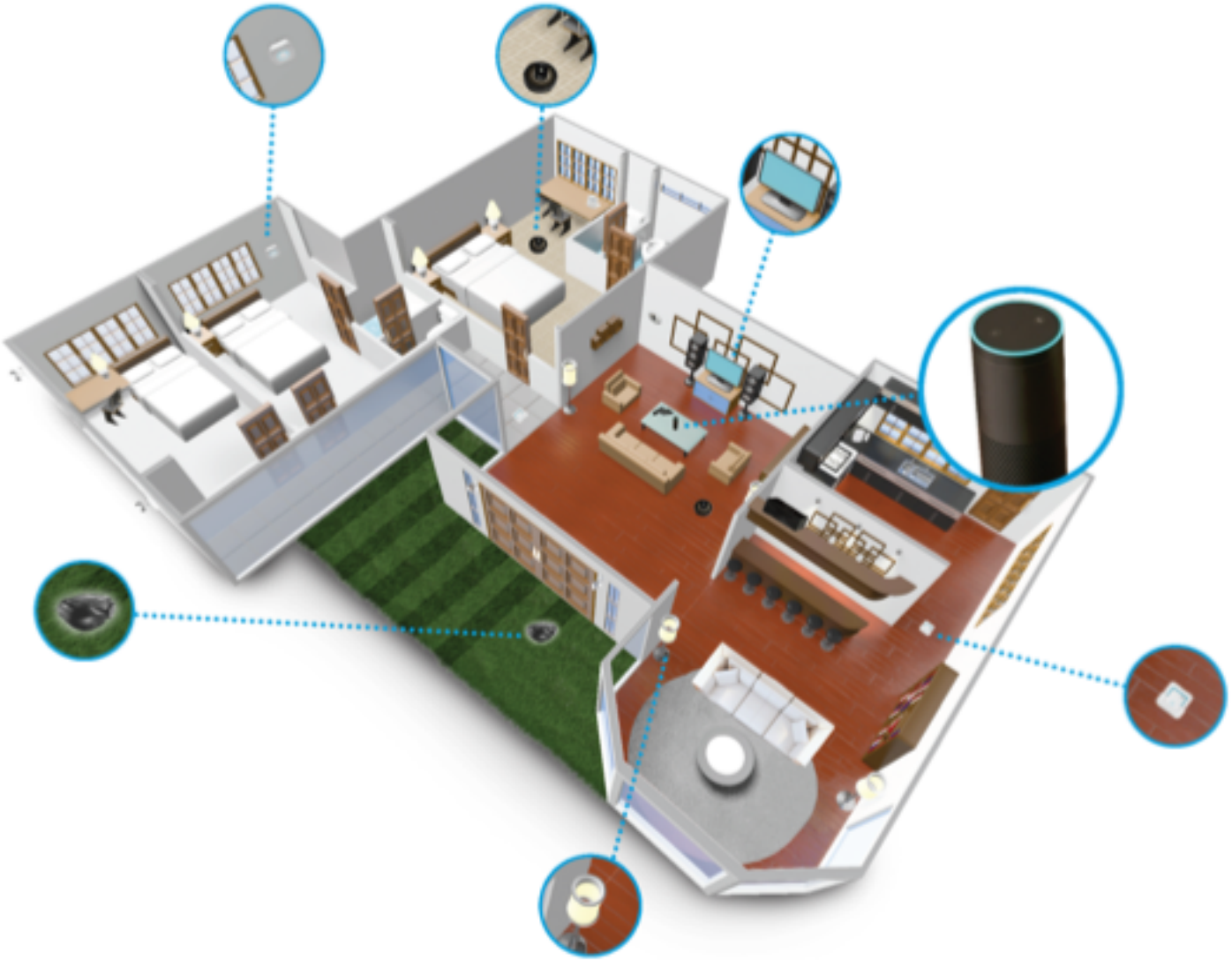
Roomba® 890

Roomba® 960

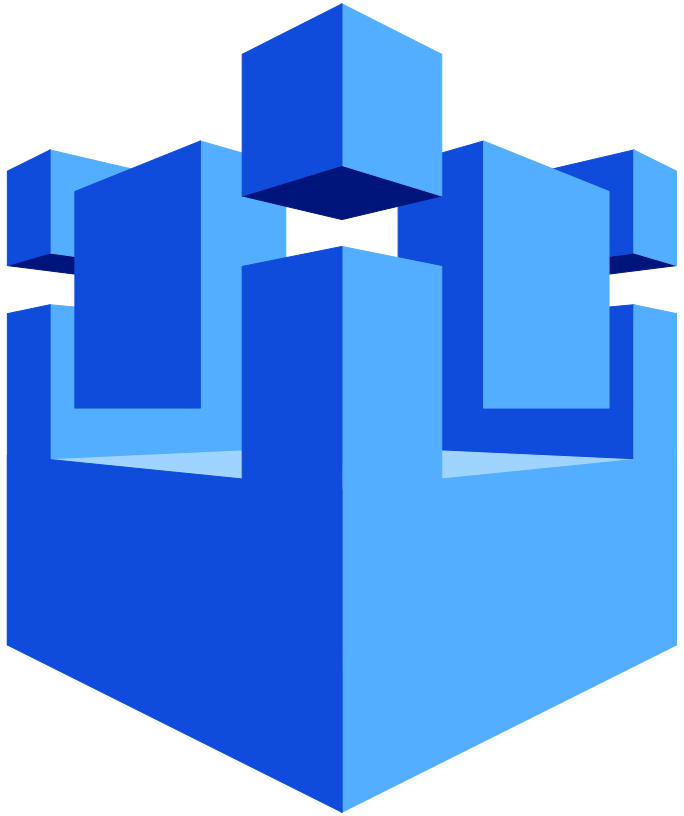
Roomba® 980

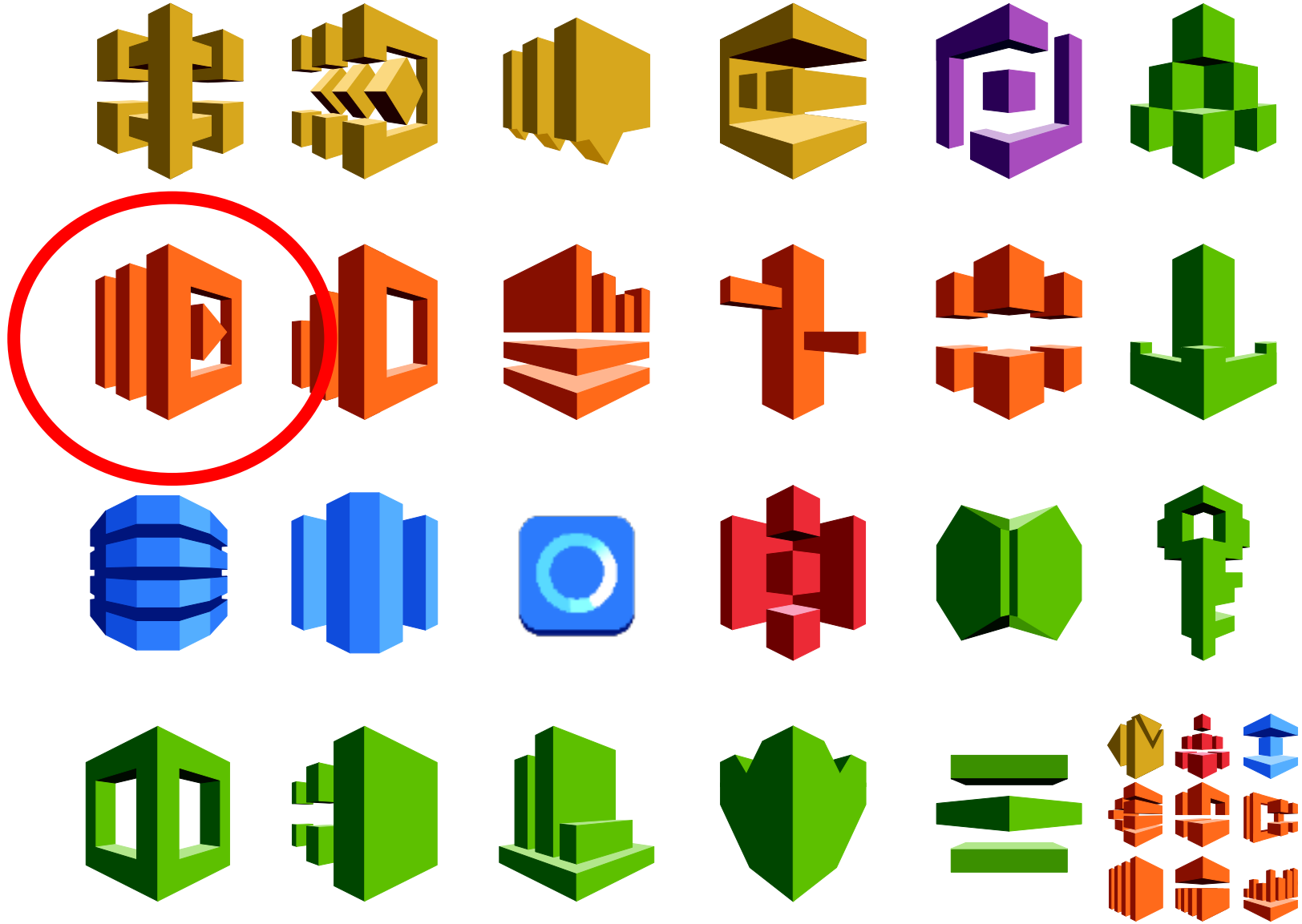
Then

Now



Then Now Future





# Why serverless?



# The reasons for serverless

**What:**

- Service-full + ephemeral compute
  - Not always F, not always aaS
- Resources billed → resources used
- Smaller, more abstract control plane

**Why:**

- Lower cost
- Lower operations burden
- Faster time to market
- Focus on business value

# Serverless architecture

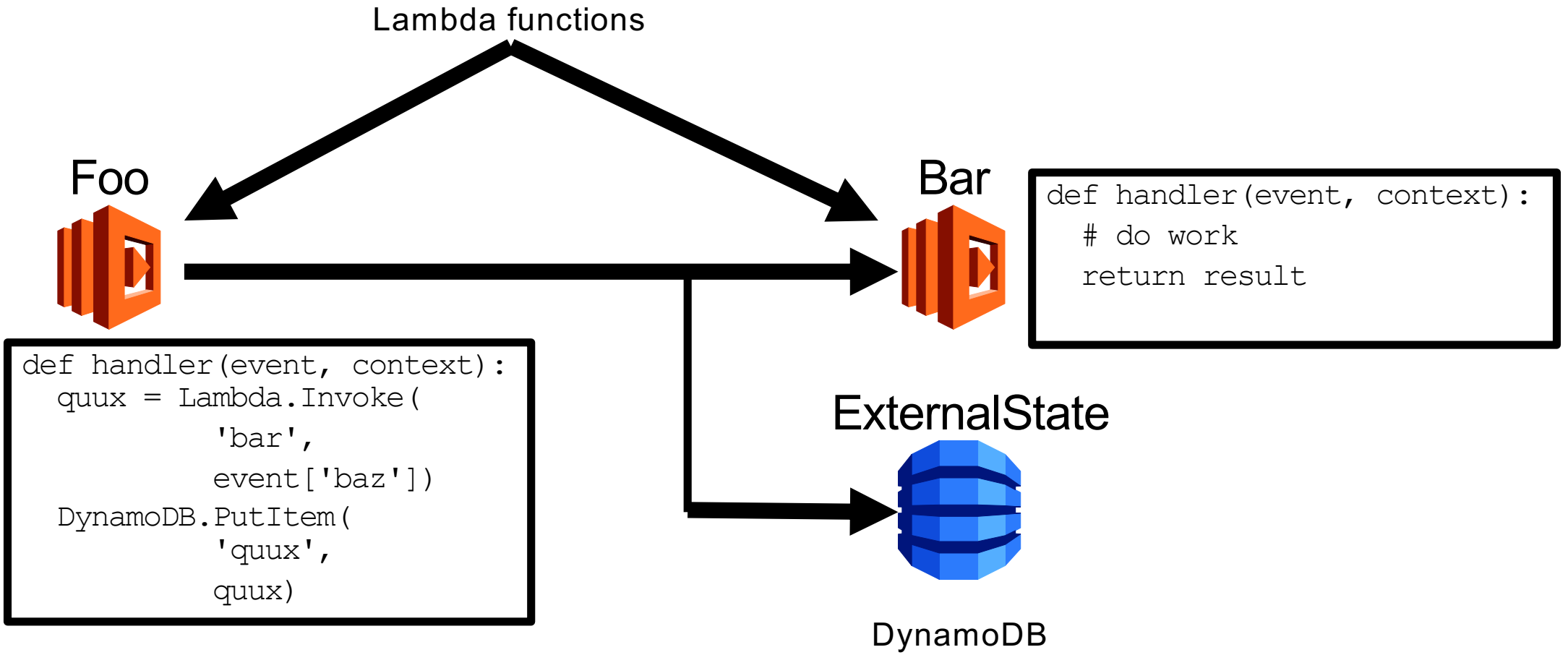
- Call graph → component graph
- Distributed system thinking
  - Traditionally occurs at system boundaries
  - Serverless: must be treated systematically
- Build robust-by-design systems

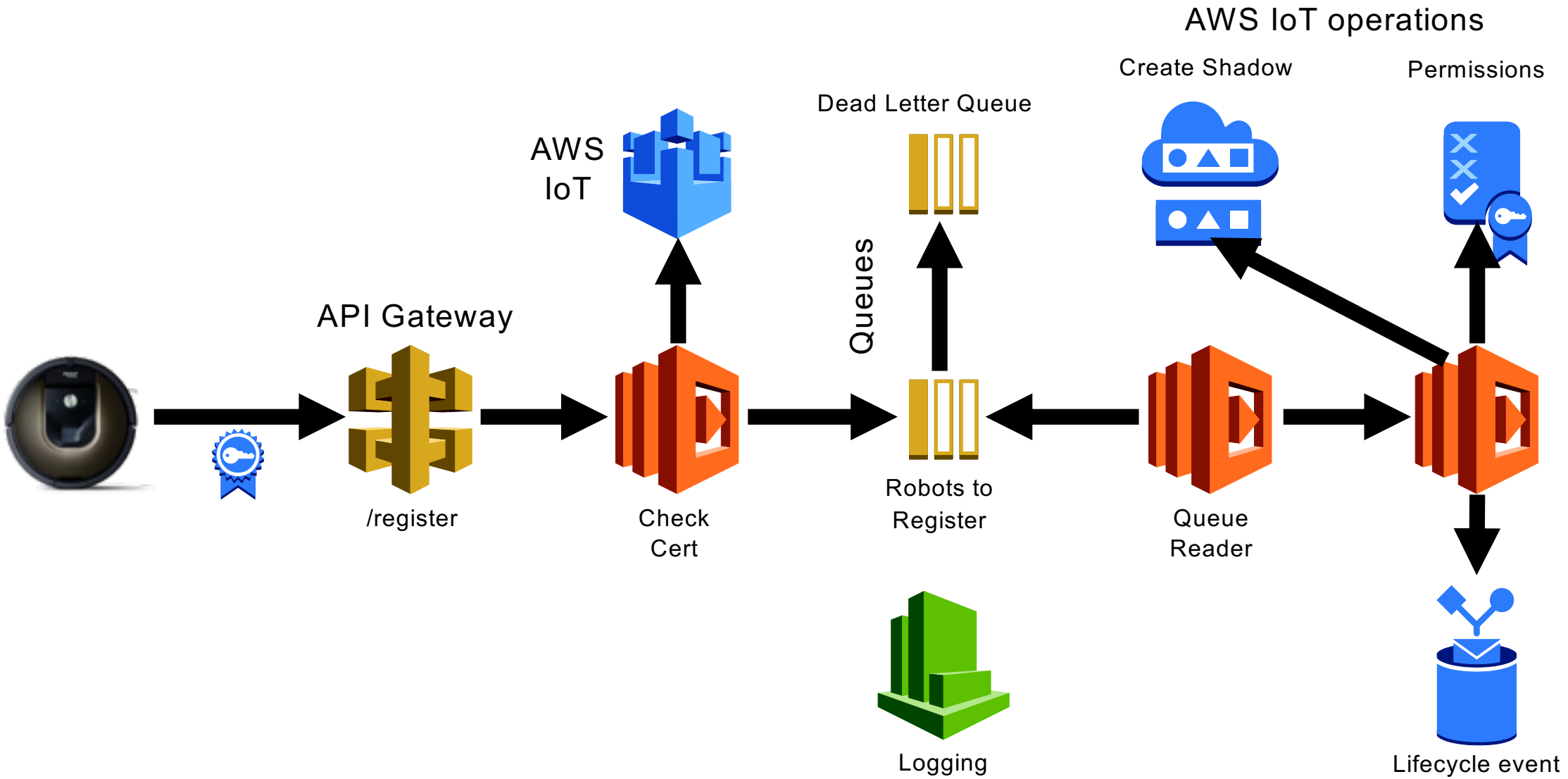
# Before serverless

## aka the dark ages

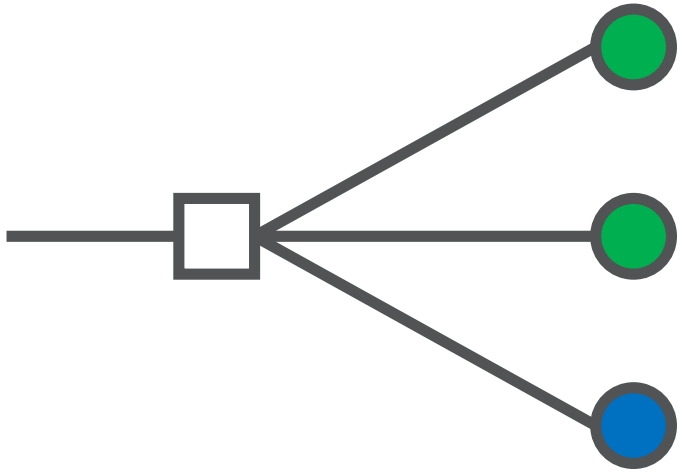
```
def foo(input):
  quux = bar(input.baz)
  internalState.quux = quux

def bar(input):
  # do work
  return result
```

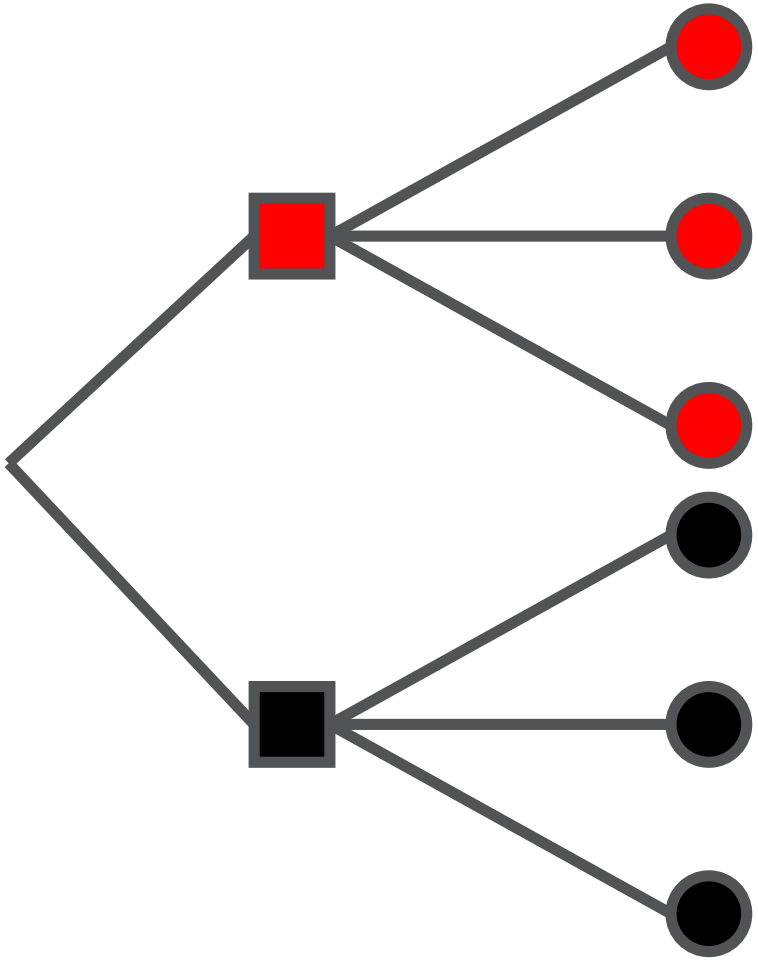
EC2


Lambda functions
Foo
Bar
def handler(event, context):
# do work
return result
def handler(event, context):
quux = Lambda.Invoke(
'bar',
event['baz'])
DynamoDB.PutItem(
'quux',
quux)
ExternalState
DynamoDB


AWS IoT operations
Create Shadow
Permissions
Dead Letter Queue
AWS IoT
Queues
API Gateway
/register
Check Cert
Robots to Register
Queue Reader
Logging
Lifecycle event

# Deployment

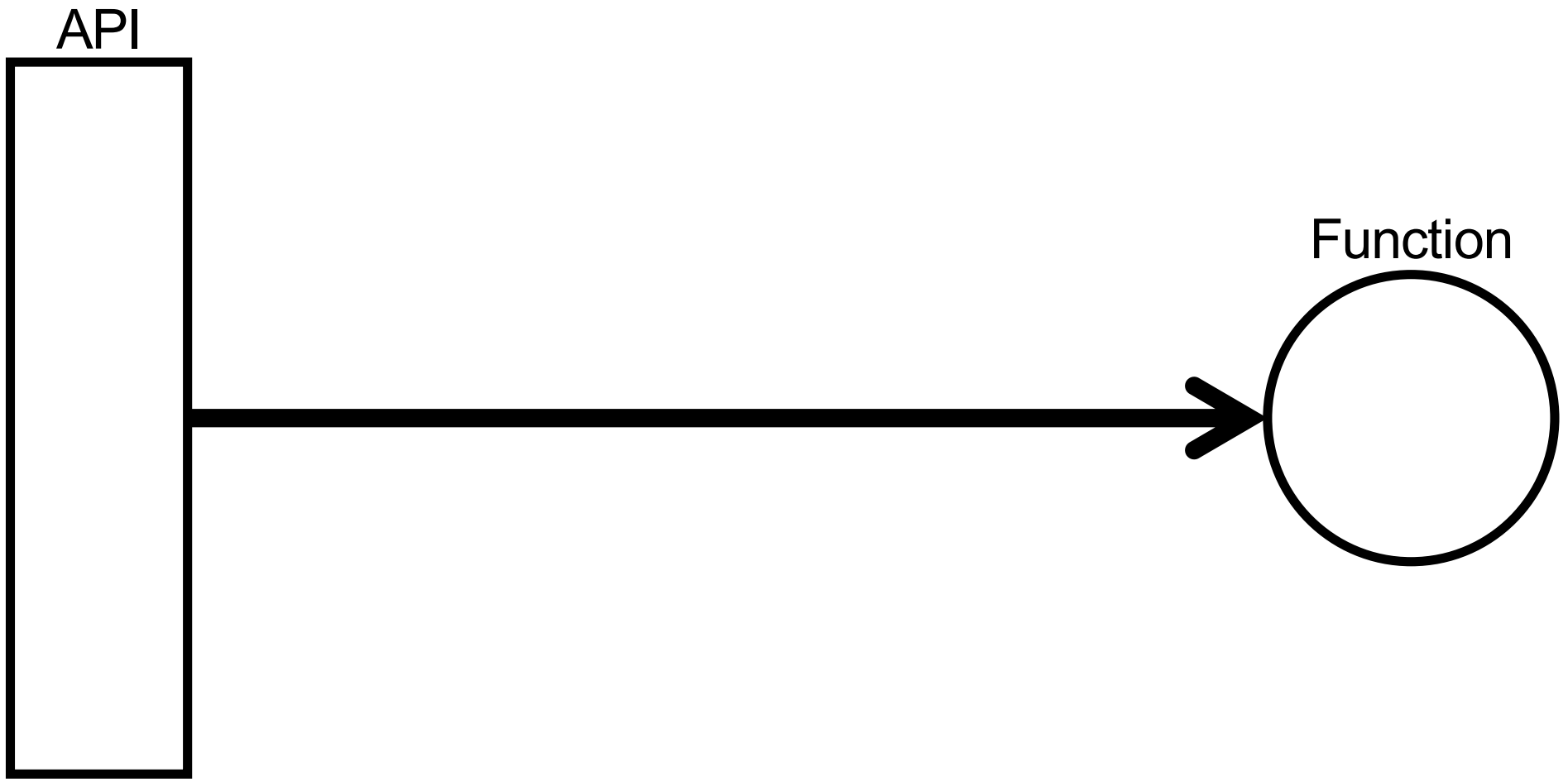


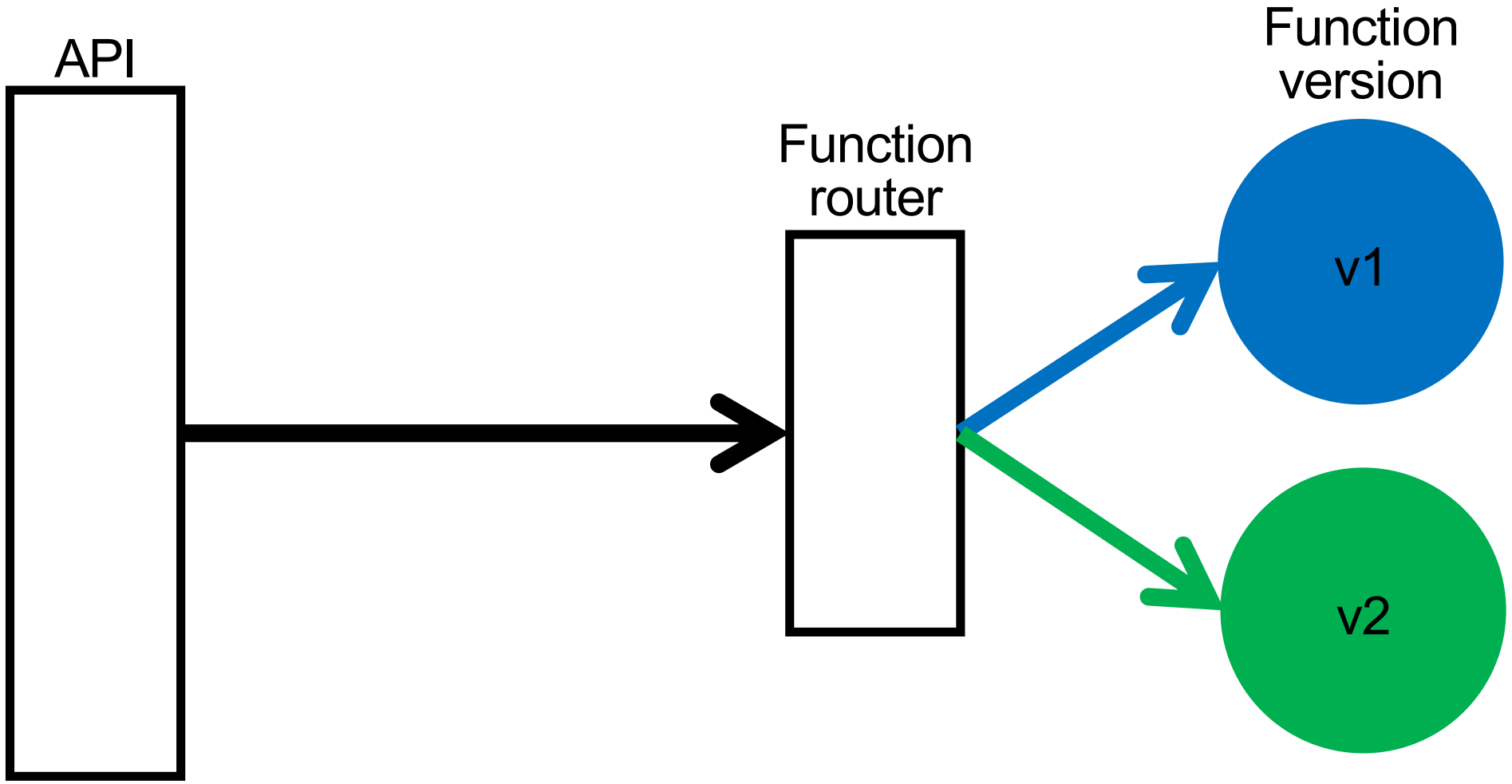


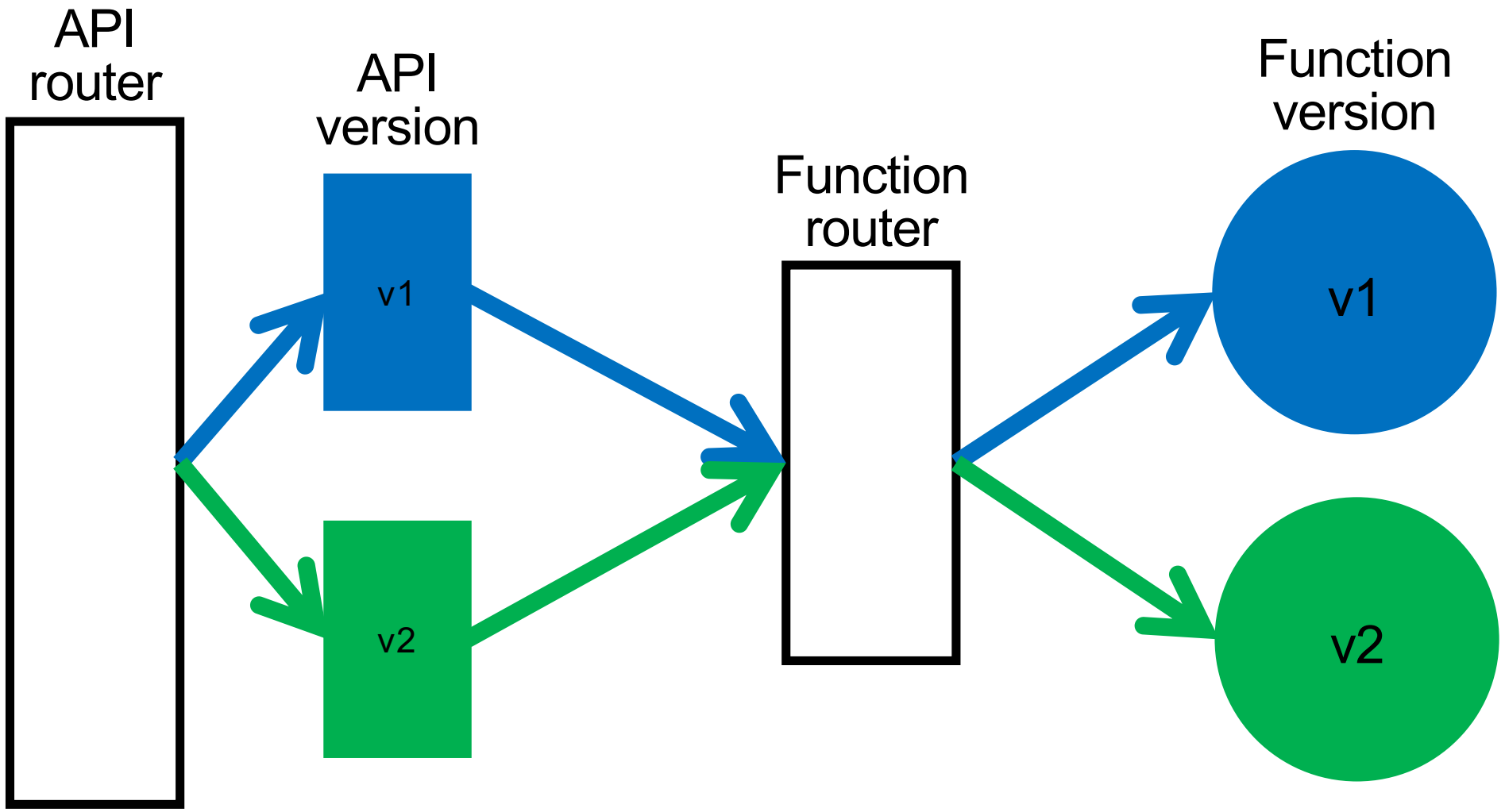
# Deployment

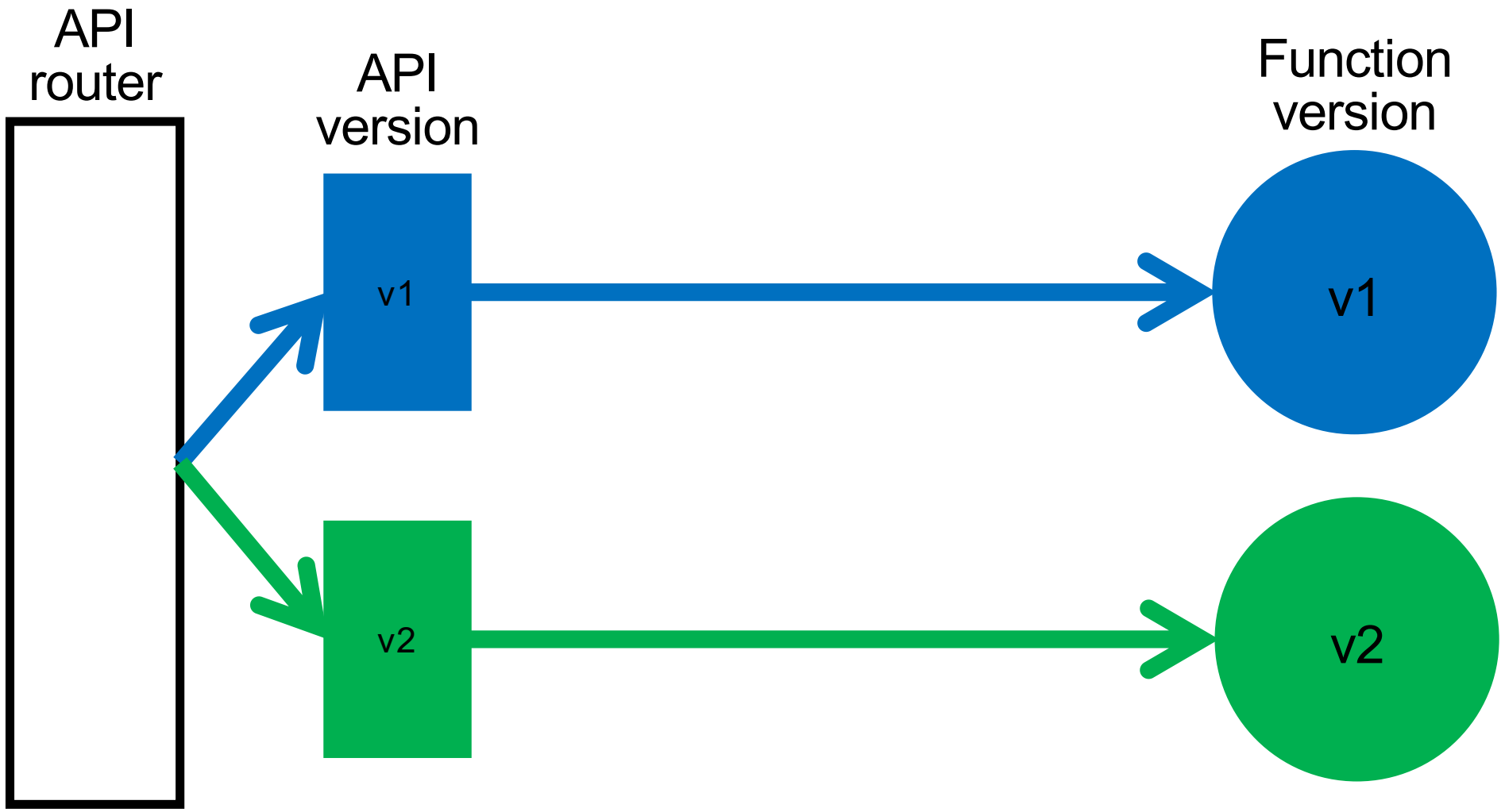
- Red/black imposes requirements on clients
- Blue/green is the direction providers are headed
- Existing paradigm:
  - Blue/green controller is part of your component graph
  - Update component graph in-place
  - Controller manages roll-out



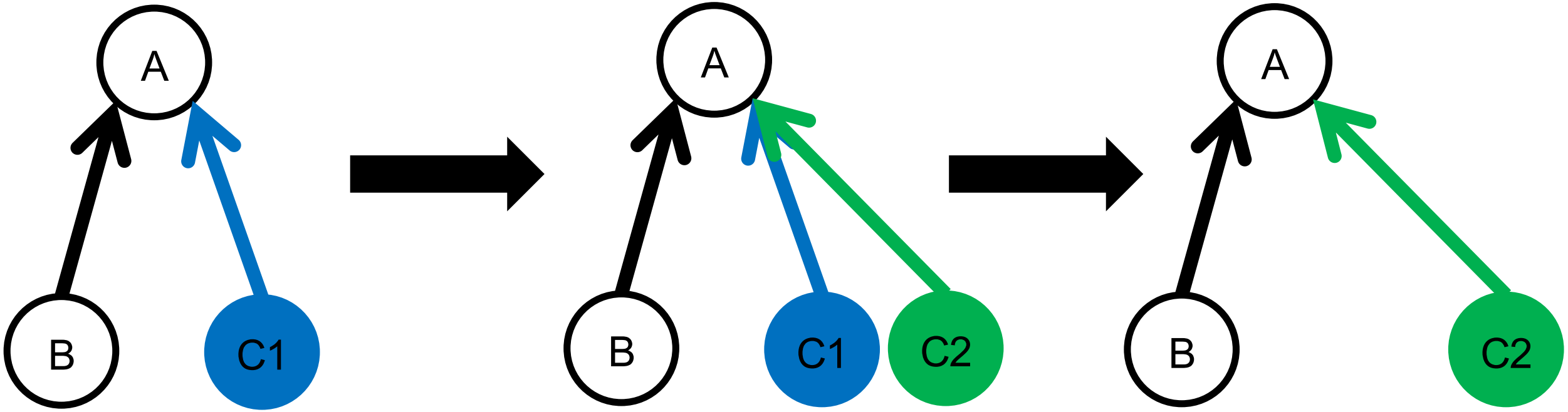

API
Function

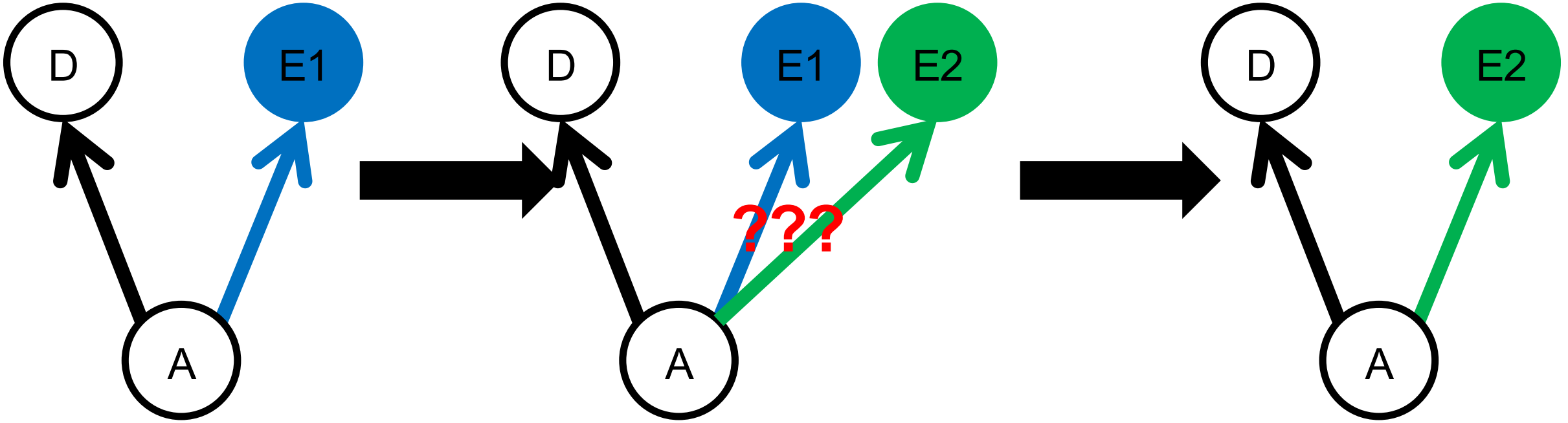

API
Function router
Function version
v1
v2

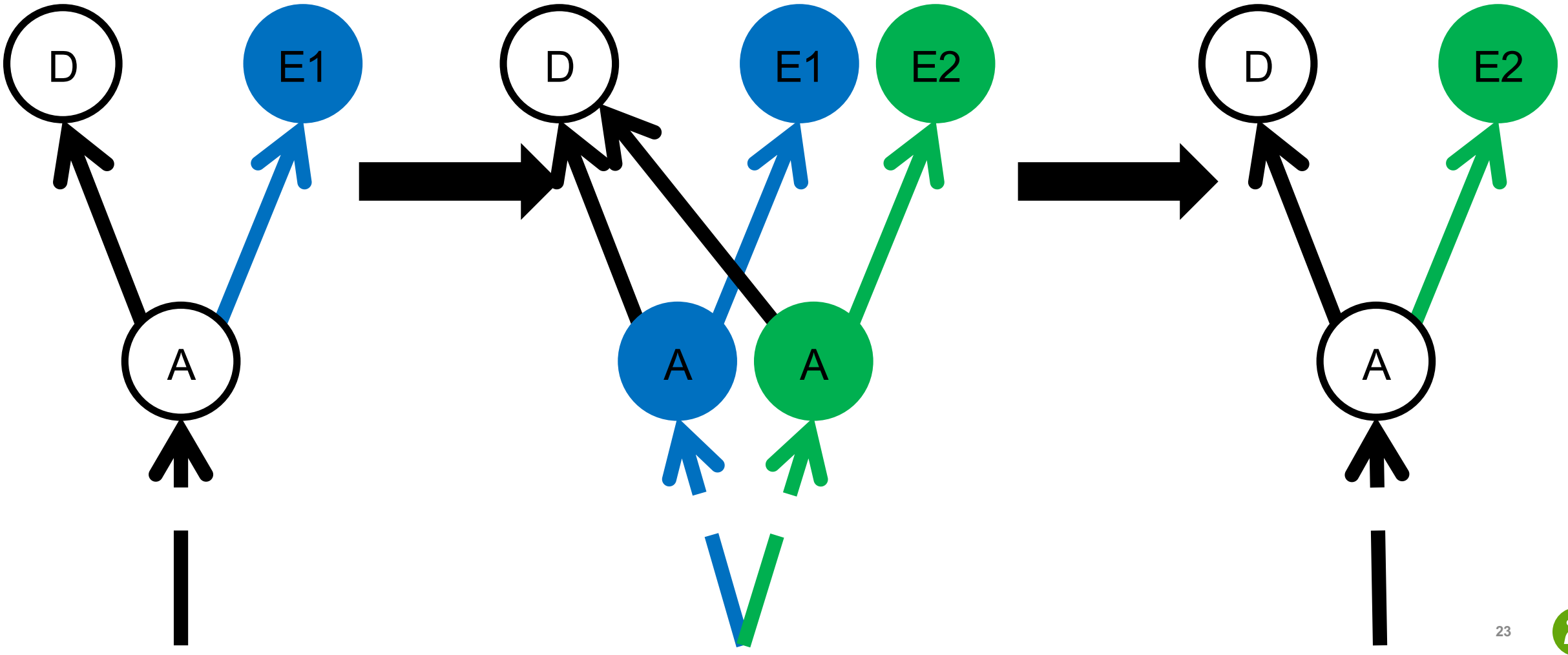

API
router
API
version
v1
v2
Function
router
Function
version
v1
v2

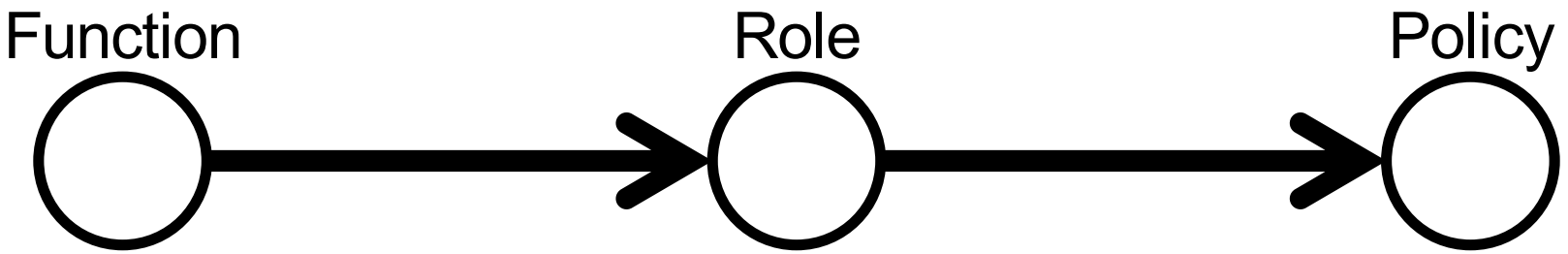


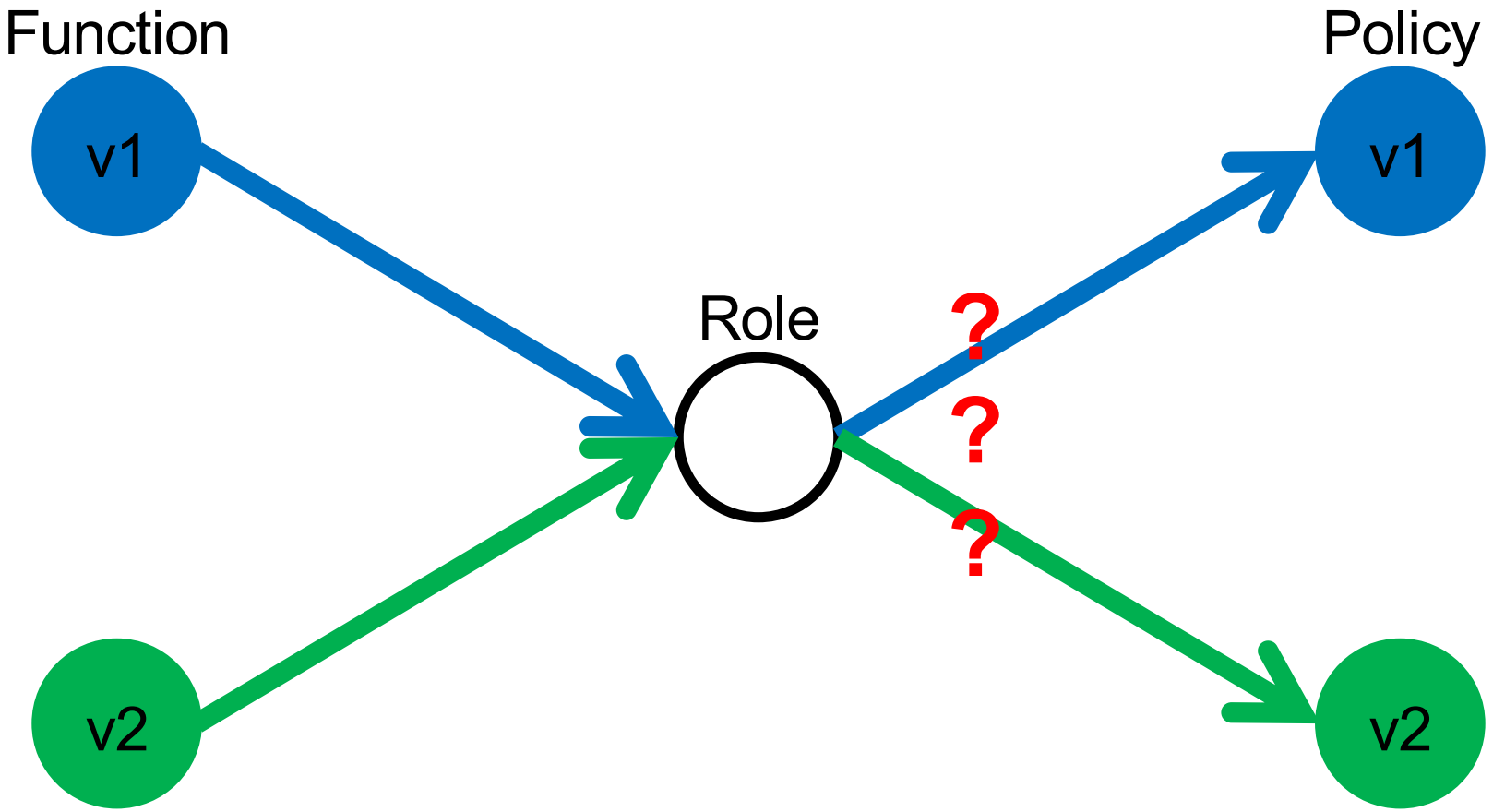

Function/code versions must be first-class citizens in infrastructure


A
B
C1
A
B
C1
C2
A
B
C2


D
E1
A
D
E1
E2
???
A
D
E2
A


D
E1
A
D
E1
E2
A
A
D
E2
A

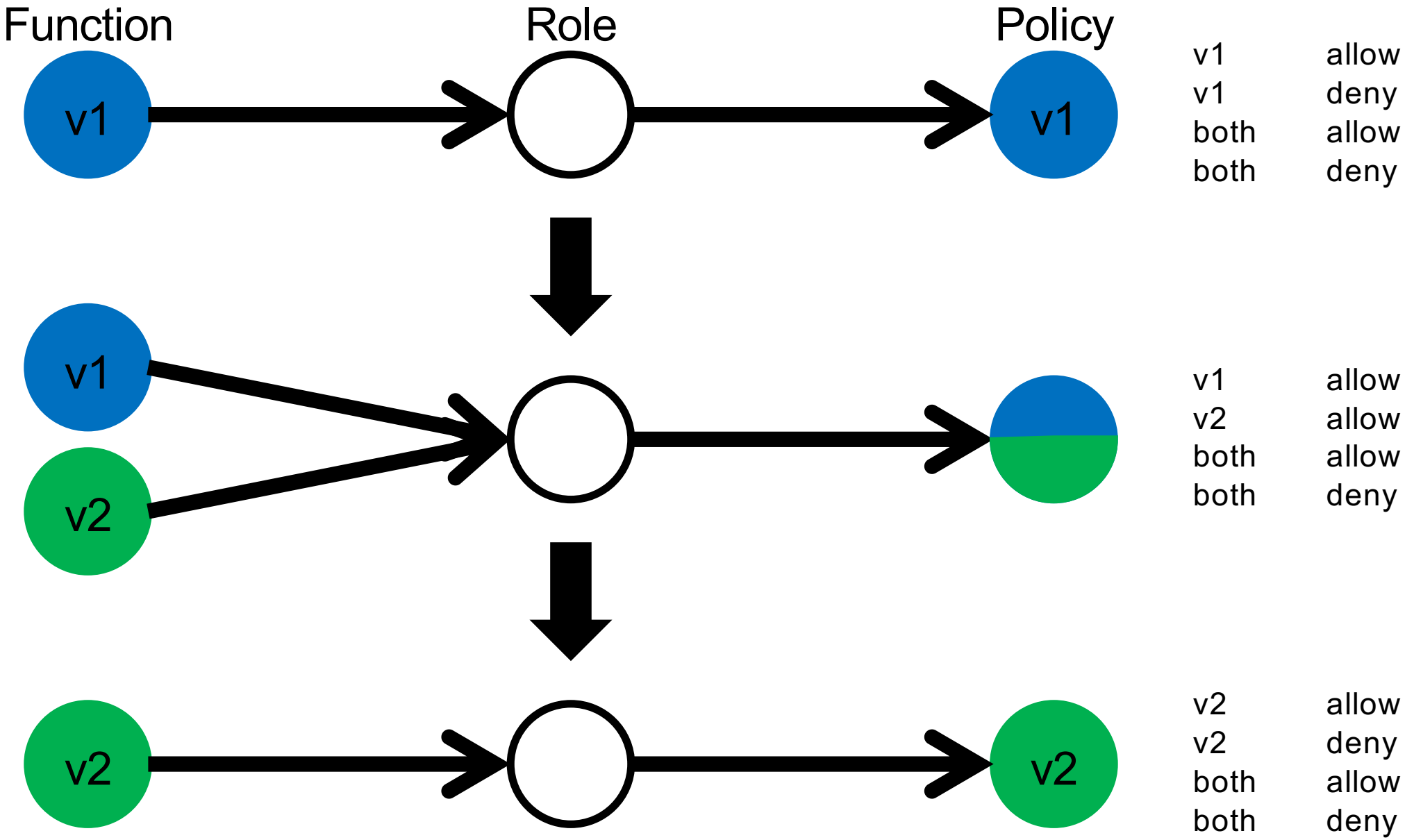

Function
Role
Policy




| | |
|---|---|
| v1 | allow |
| v1 | deny |
| both | allow |
| both | deny |

| | |
|---|---|
| v2 | allow |
| v2 | deny |
| both | allow |
| both | deny |

Continuity of role may be necessary


Function
Role
Policy
v1
v1
v1
v2
v2
v2
v1 allow
v1 deny
both allow
both deny
v1 allow
v2 allow
both allow
both deny
v2 allow
v2 deny
both allow
both deny

# Summing up

- Red/black is actually pretty easy for serverless, but harder on the service consumers
- Blue/green component graph gets *complicated*
- Existing deployment tools don't represent intermediate states
- Native support for blue/green component graphs is critical



# Questions?

iRobot