NSQ

realtime distributed message processing at scale

https://github.com/bitly/nsq

November 8th 2012 - NYC Golang Meetup

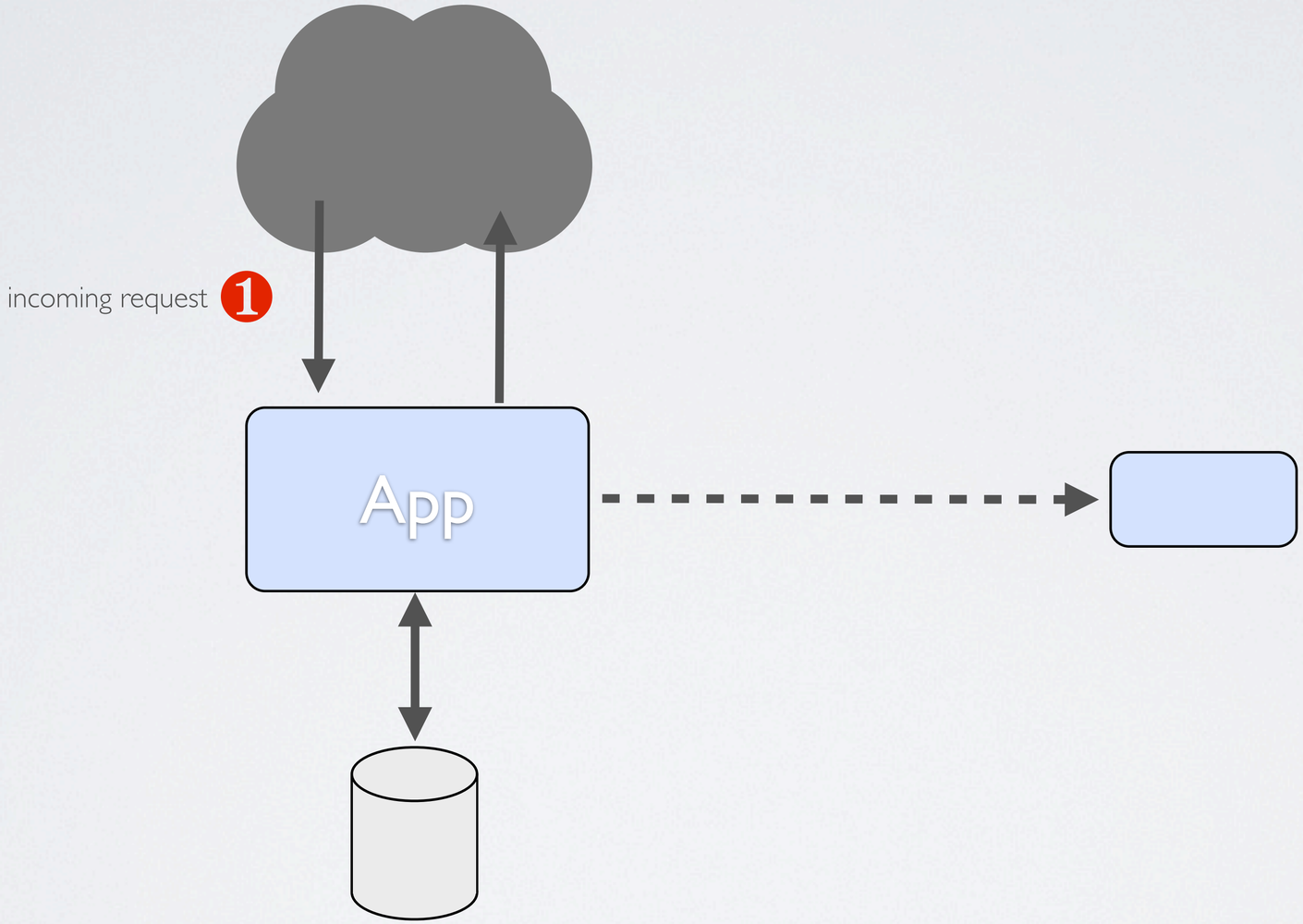
@imsnakes & **@jehiah** (infrastructure **@bitly**)

THE WAY OF THE BITLY

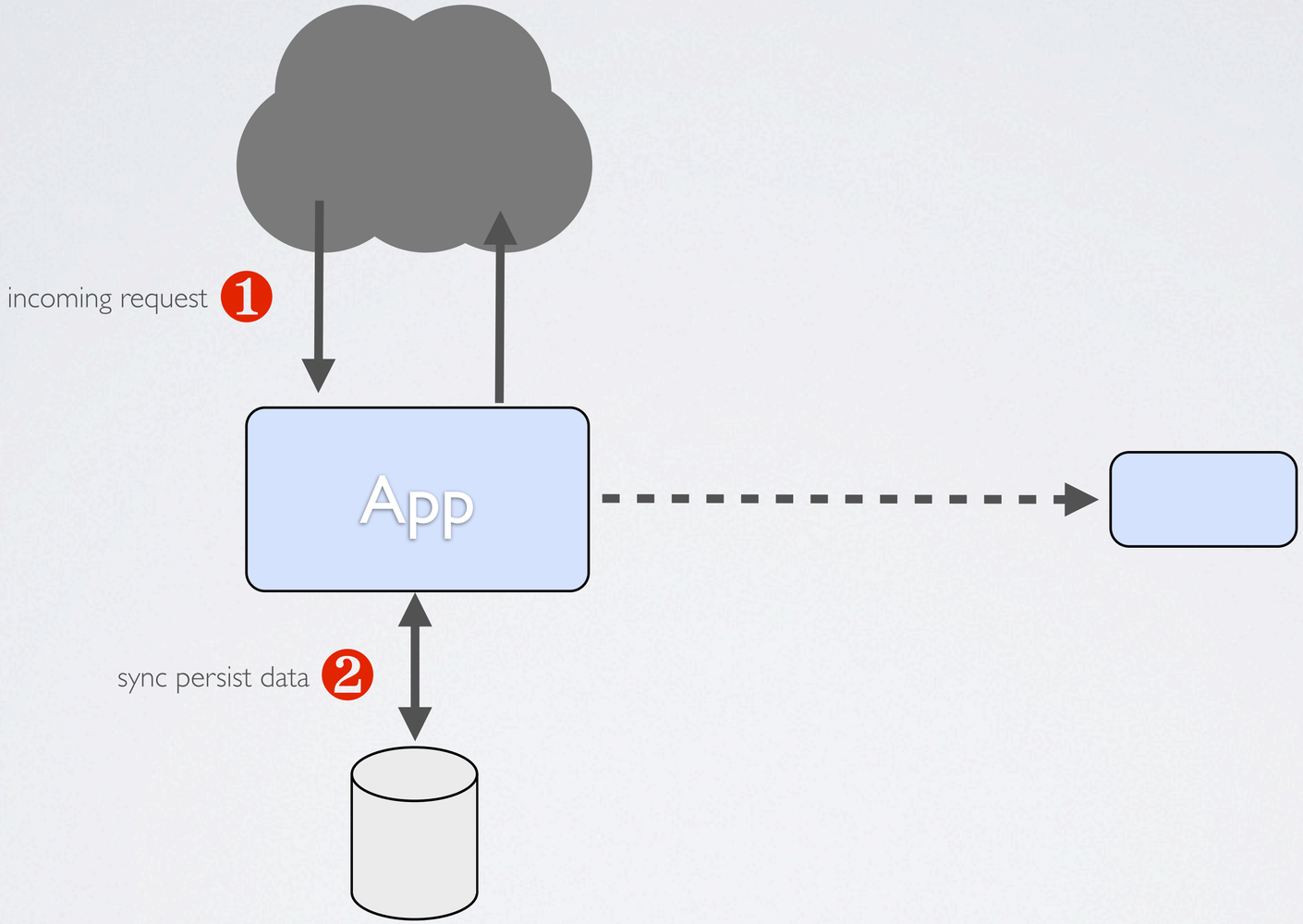
PHILOSOPHY

- service-oriented
- perform work asynchronously
- queue messages (locally)
- workers process messages (aka "queuereader")
- scale # of workers (and backend) based on ability to handle message volume
- dependencies suck (make it easy to deploy)
- use HTTP and JSON

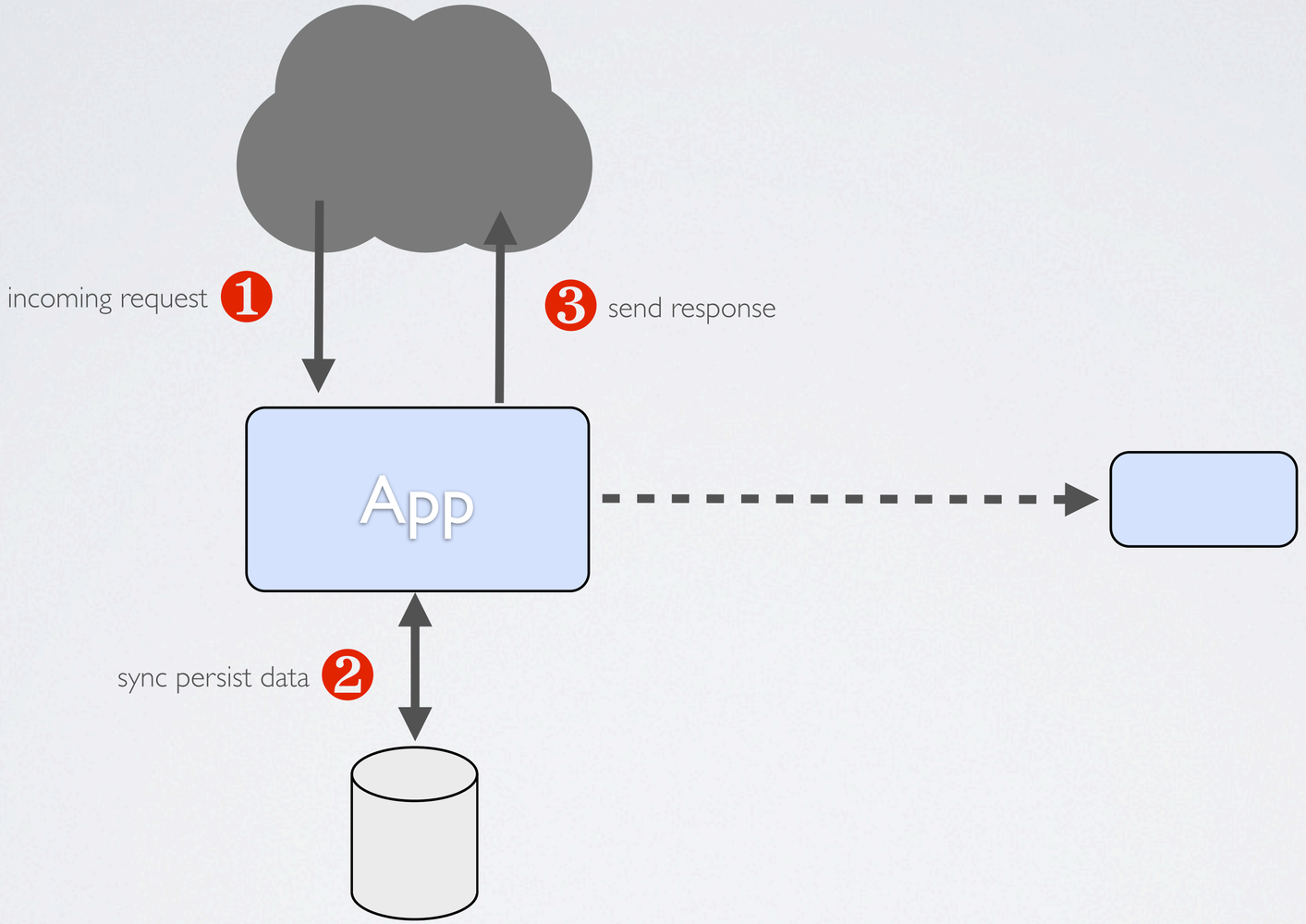
DATA FLOW



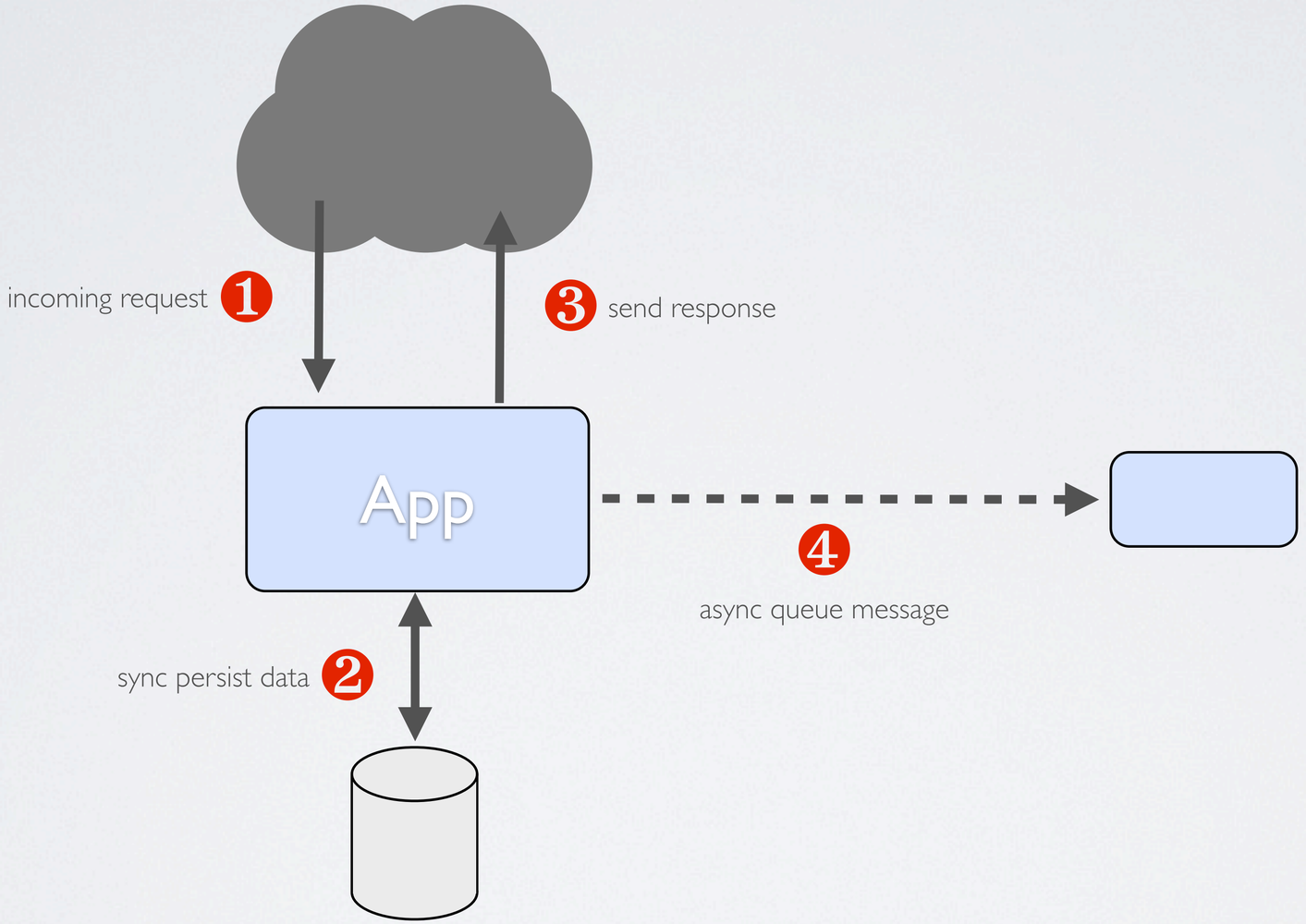
DATA FLOW



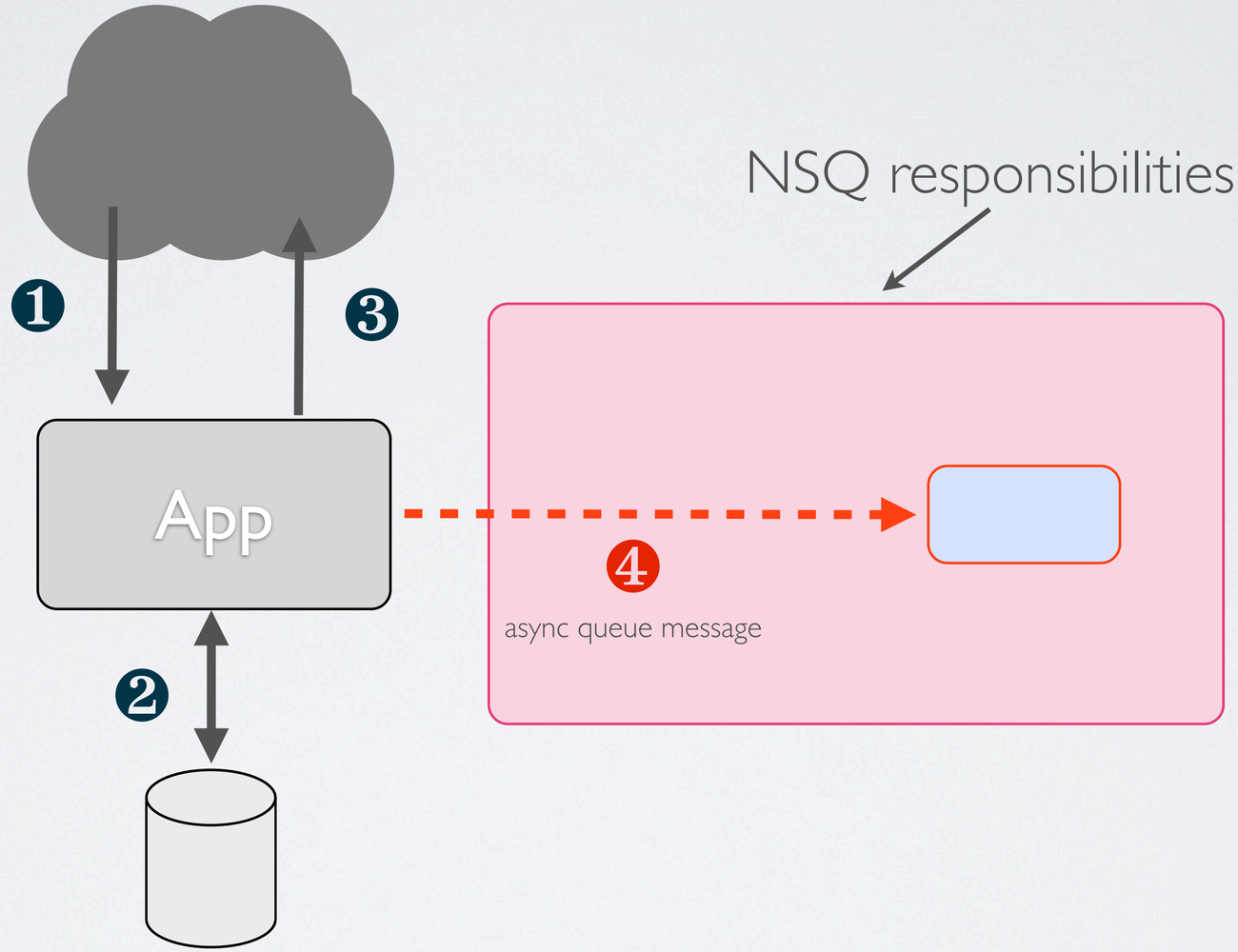
DATA FLOW



DATA FLOW


incoming request
1
3
send response
App
4
async queue message
sync persist data
2

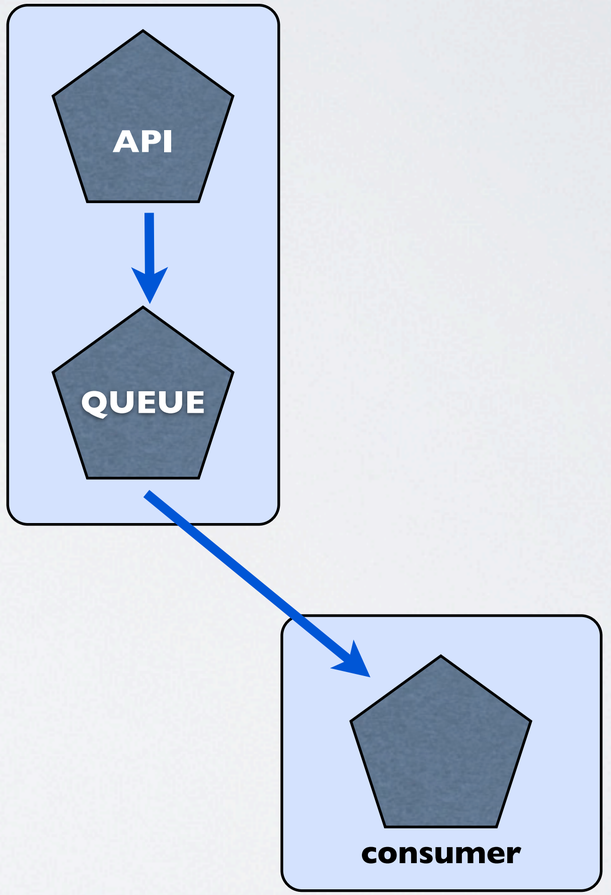
DATA FLOW



WHY QUEUE?

because things break

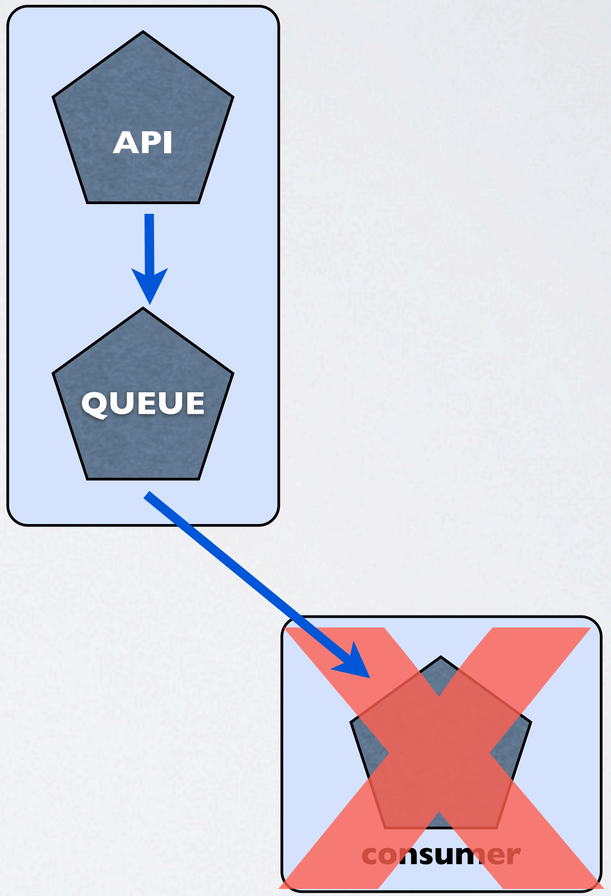
- try to avoid SPOFs
- queue(s) and worker(s) are silo'd
- in failure scenarios:
  - queues provide buffering
  - workers exponentially back off
  - messages are retried
  - aka no data loss



WHY QUEUE?

because things break

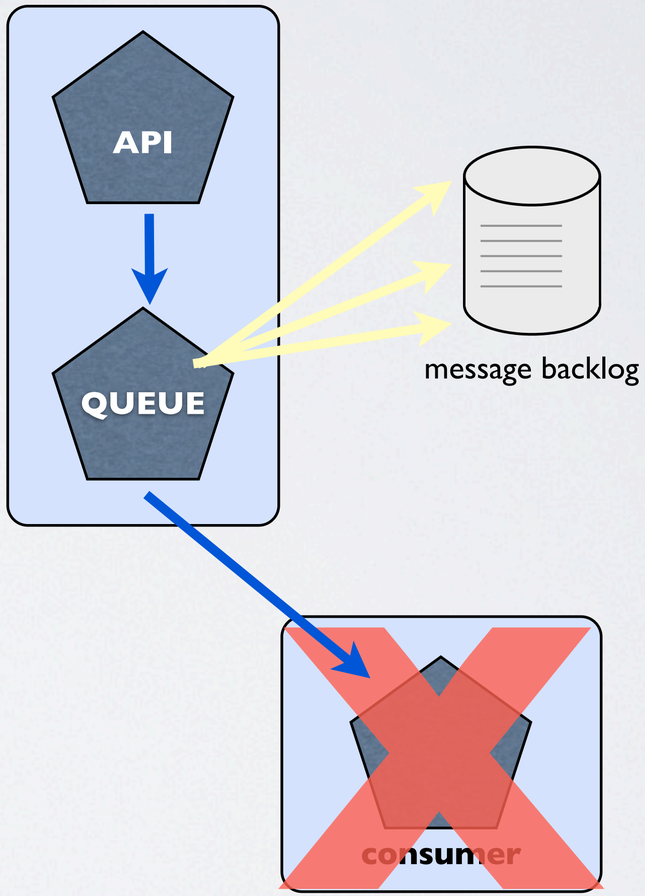
- try to avoid SPOFs
- queue(s) and worker(s) are silo'd
- in failure scenarios:
  - queues provide buffering
  - workers exponentially back off
  - messages are retried
  - aka no data loss



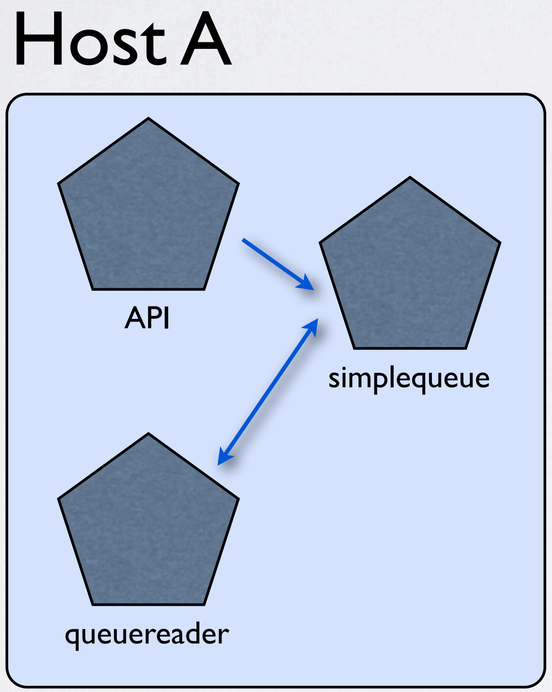
WHY QUEUE?

because things break

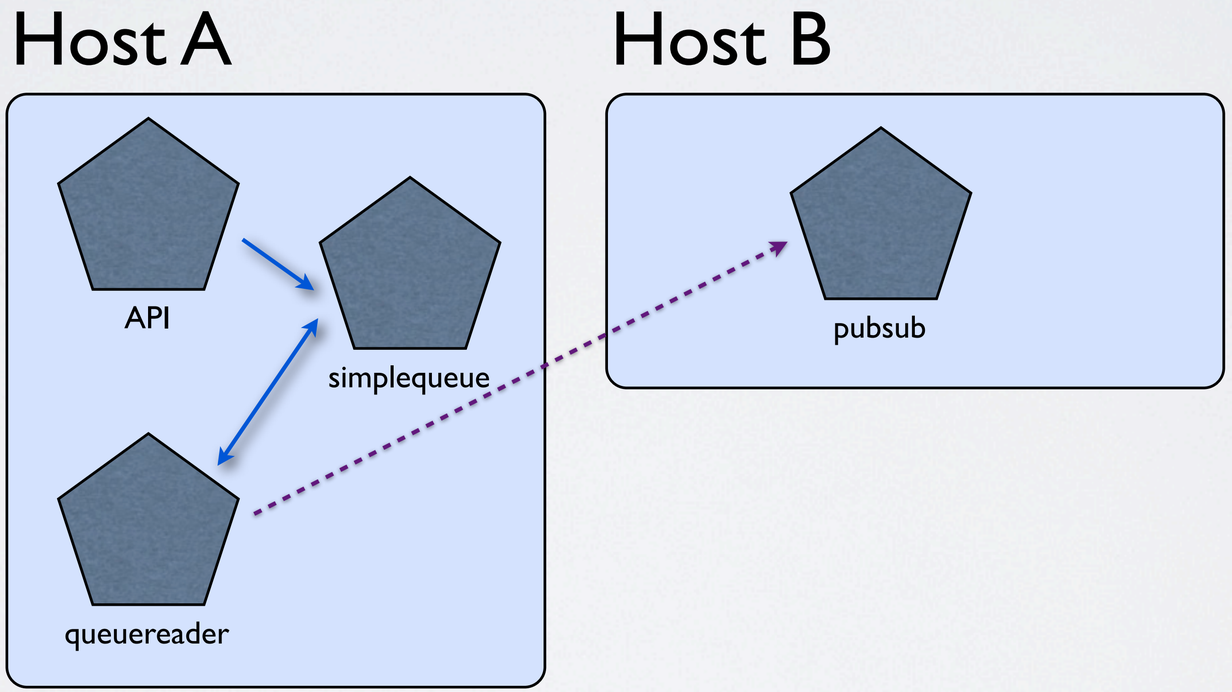
- try to avoid SPOFs
- queue(s) and worker(s) are silo'd
- in failure scenarios:
  - queues provide buffering
  - workers exponentially back off
  - messages are retried
  - aka no data loss



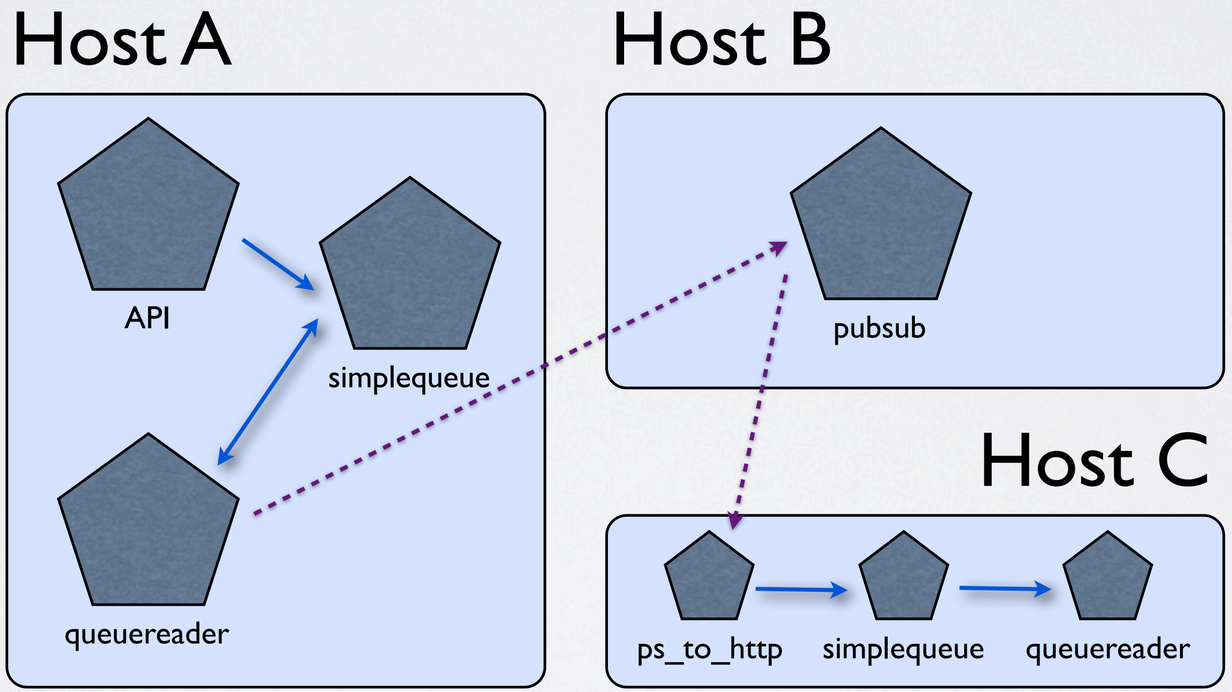
TYPICAL (OLD) ARCHITECTURE



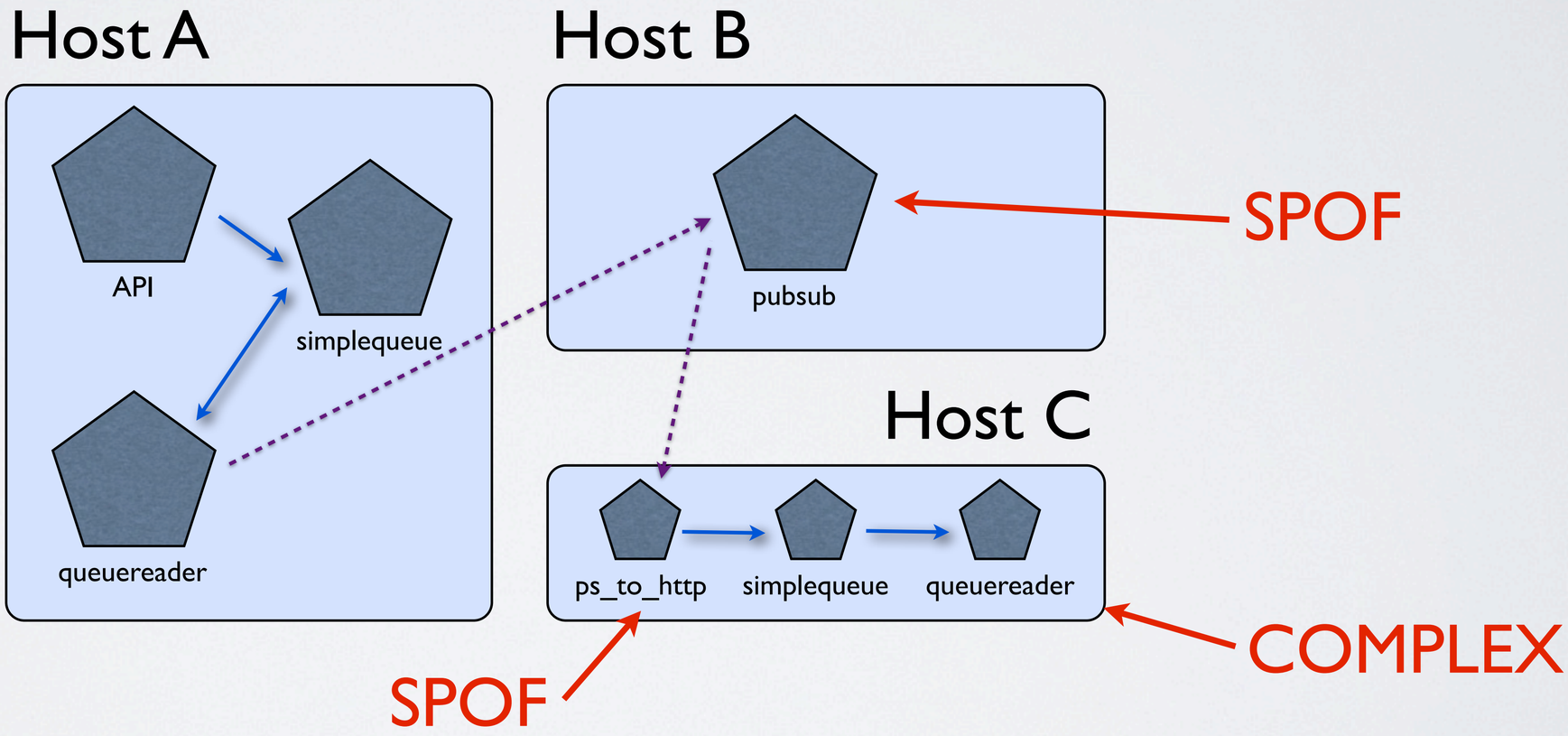
TYPICAL (OLD) ARCHITECTURE



TYPICAL (OLD) ARCHITECTURE



TYPICAL (OLD) ARCHITECTURE



THE BAD

- no message guarantees and clients are responsible for re-queueing
- bottleneck and SPOF transport issue (reconnects, distribution, fault tolerance, etc.)
- inefficiency - we copy streams multiple times to multiple systems
- complicated service setup repeated for *each* stream
- hard-coded knowledge of queue addresses in queuereaders
- lack of internal performance metrics (clients, rate, etc.)

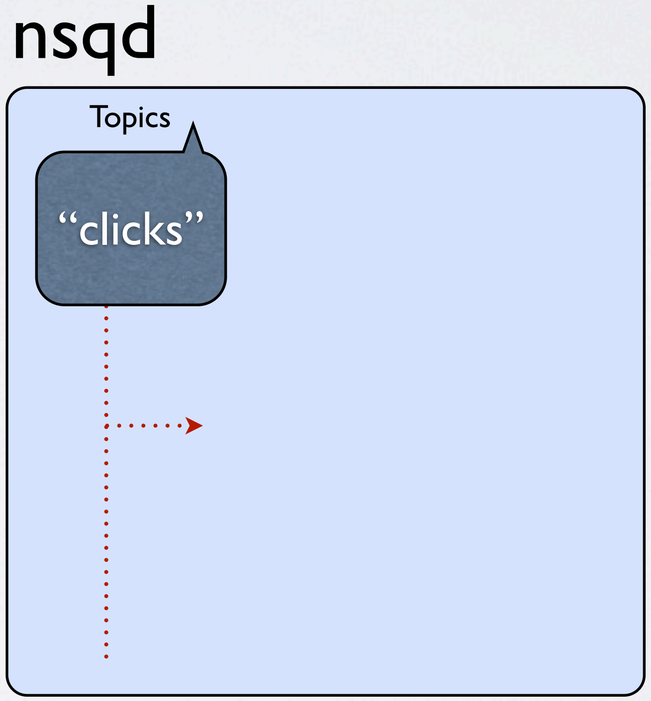
DESIGNING A SOLUTION

GOALS

- provide a straightforward upgrade path
- greatly simplify configuration requirements
- provide easy topology solutions that enable high-availability and eliminate SPOFs
- address the need for stronger message delivery guarantees
- bound the memory footprint of a single process
- improve efficiency
- out of the box library support for Go and Python

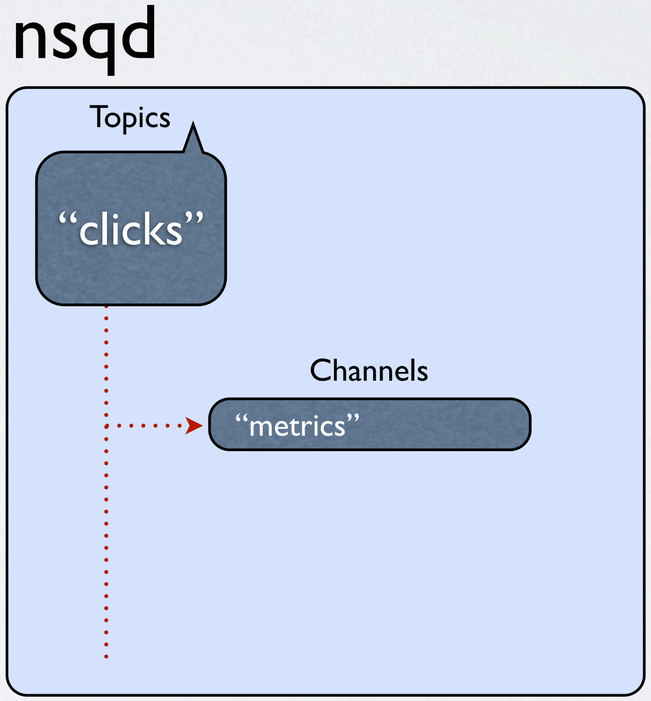
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



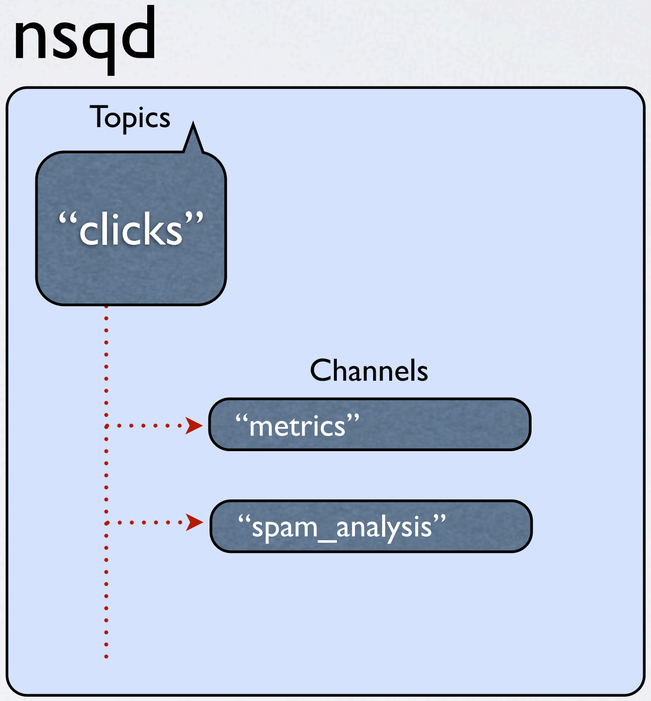
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



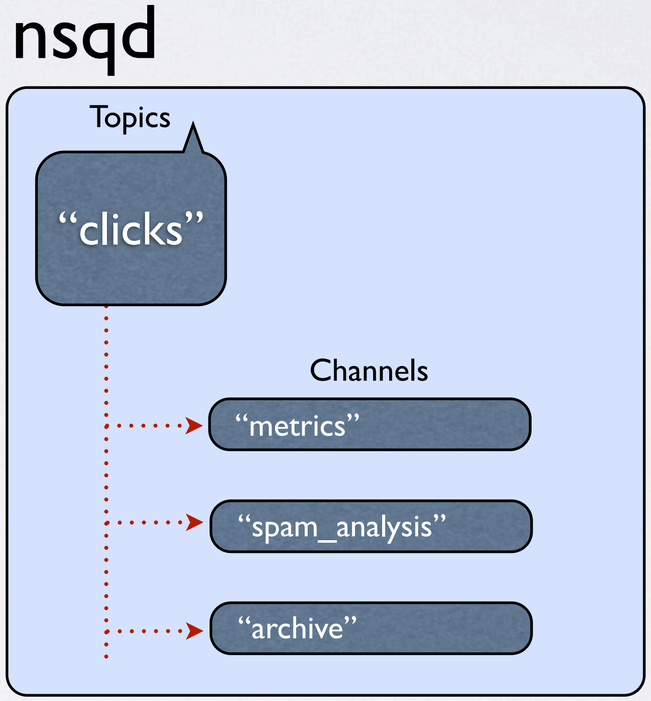
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



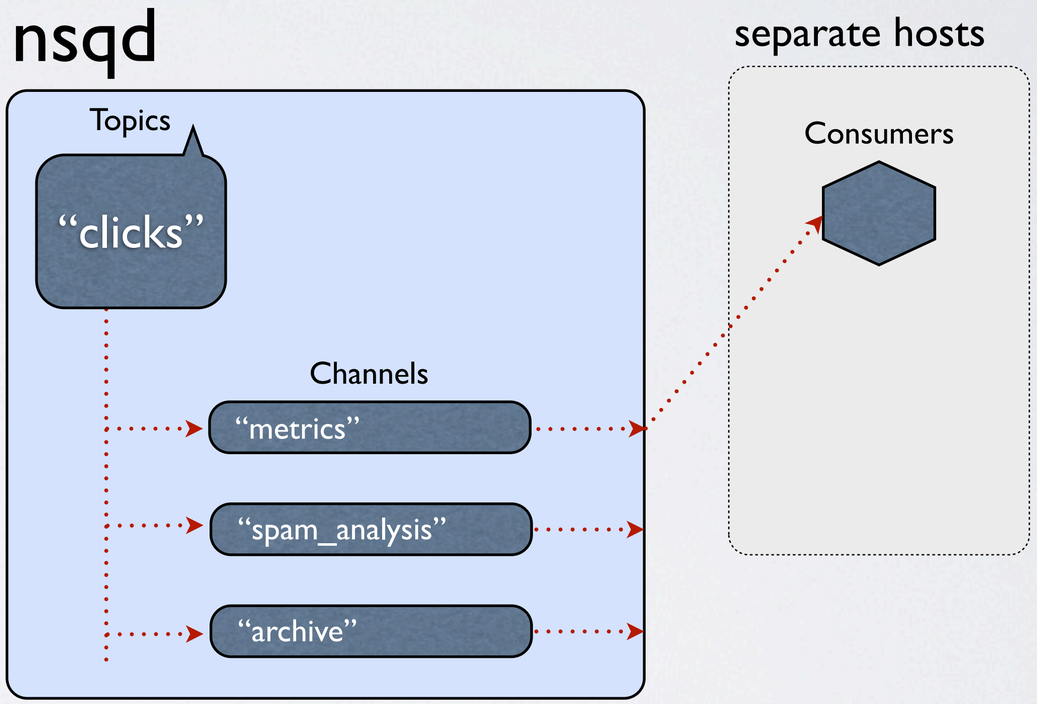
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



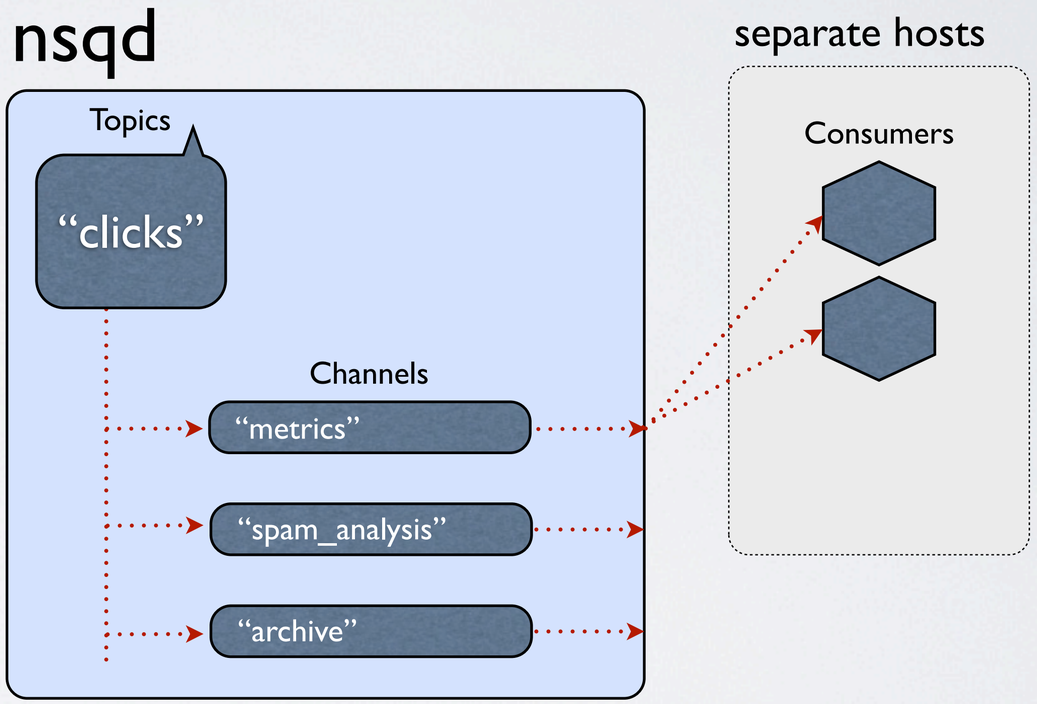
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



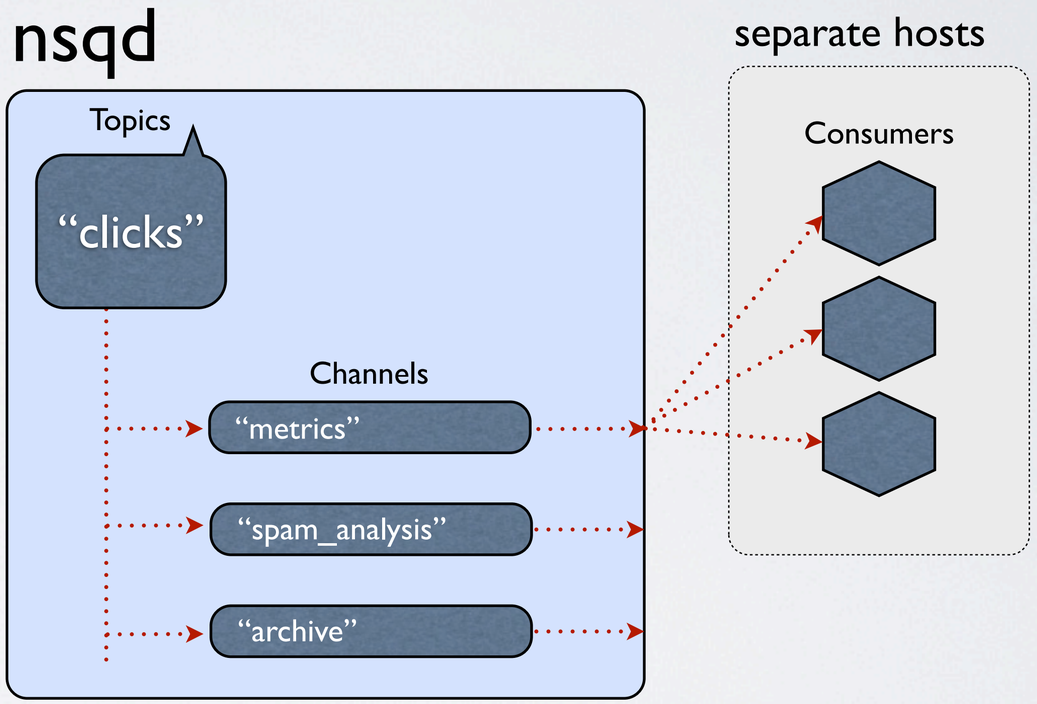
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



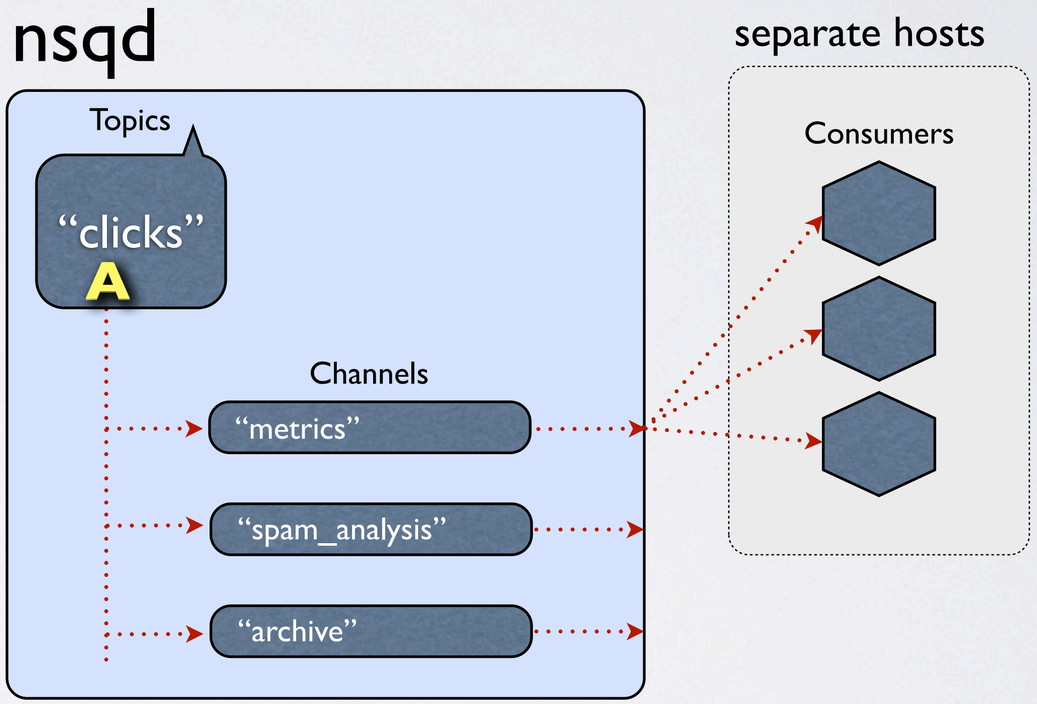
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



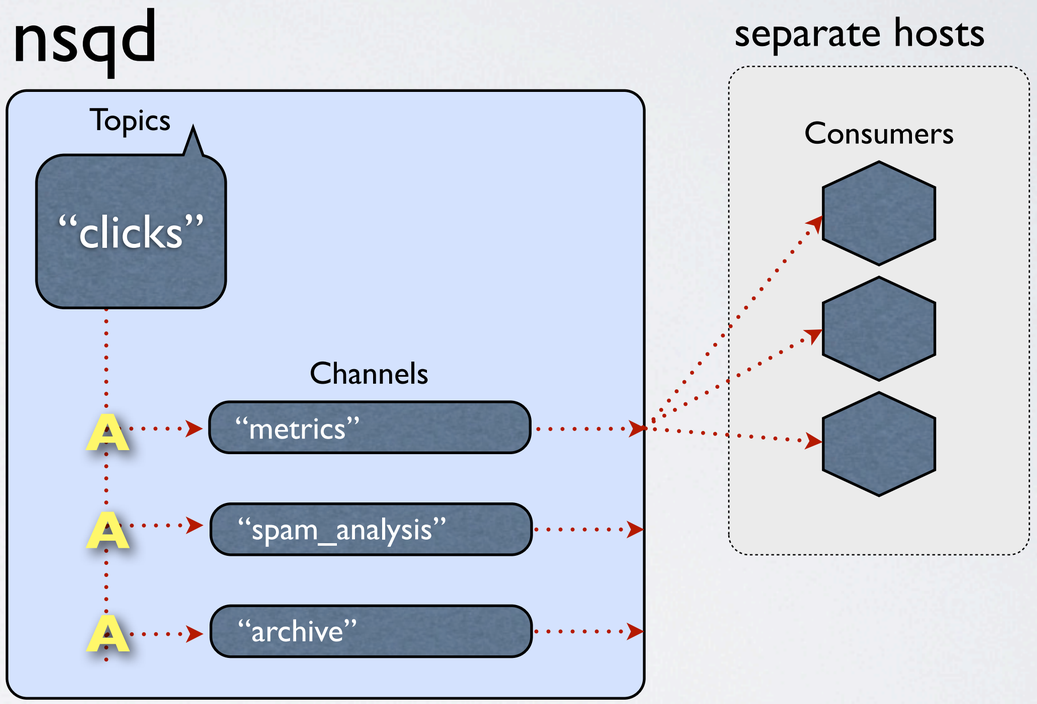
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



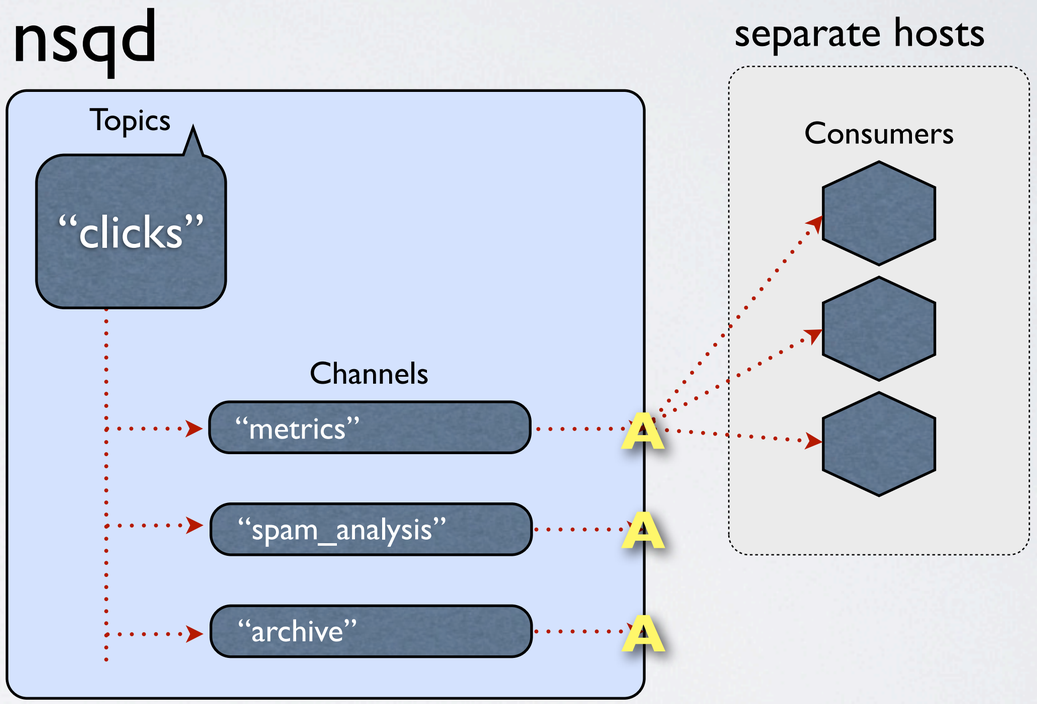
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



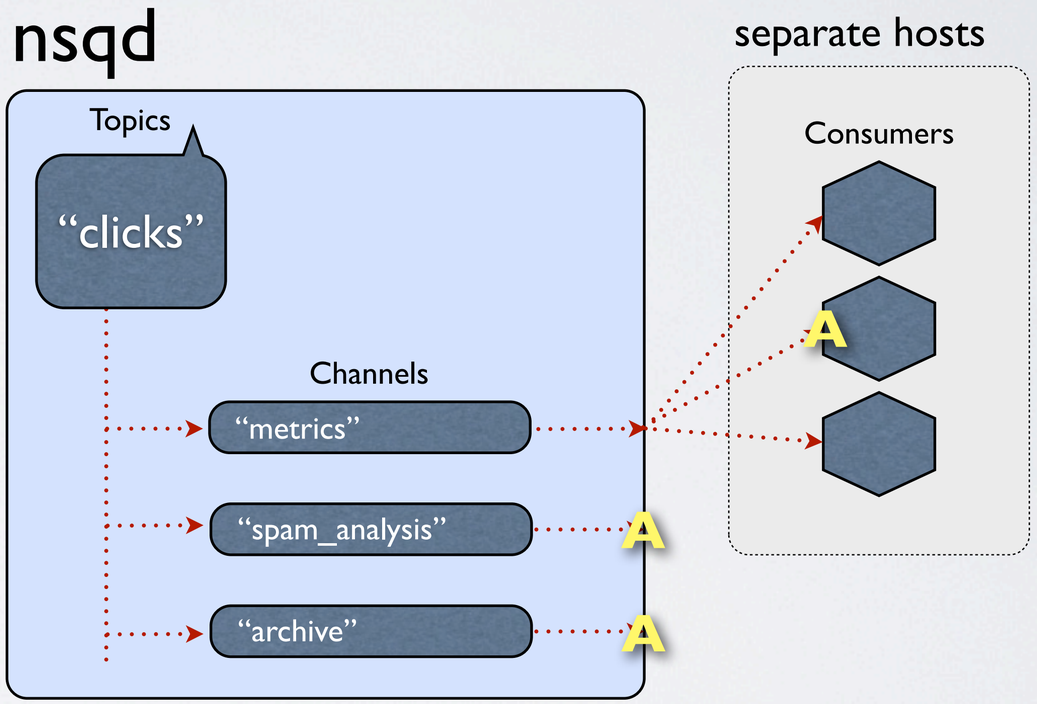
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



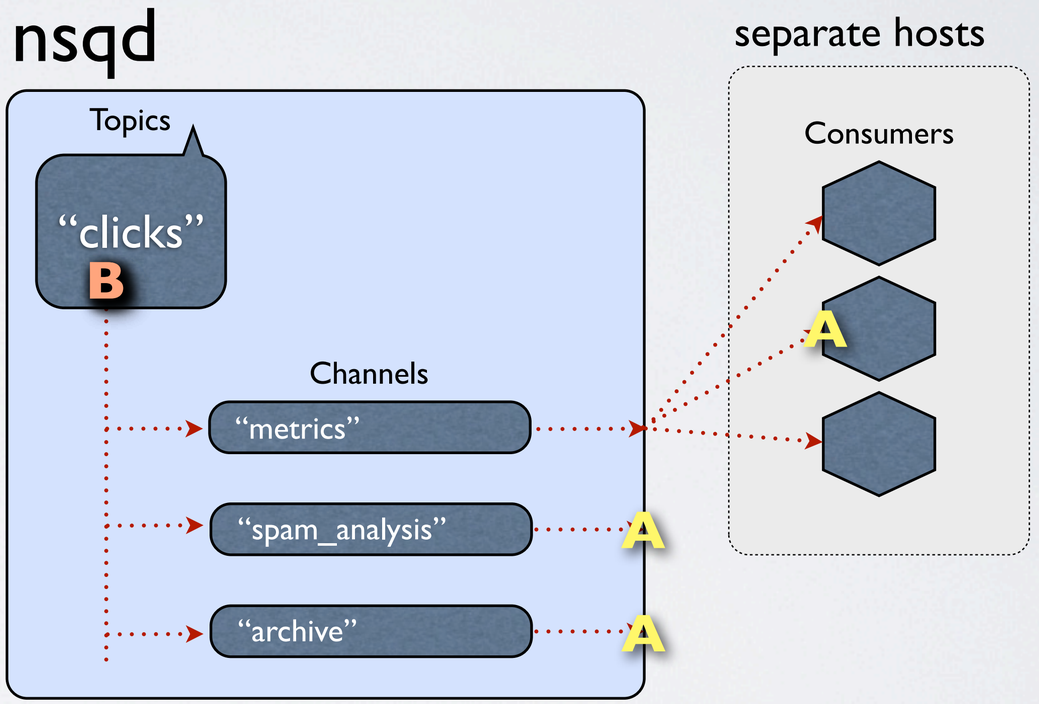
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



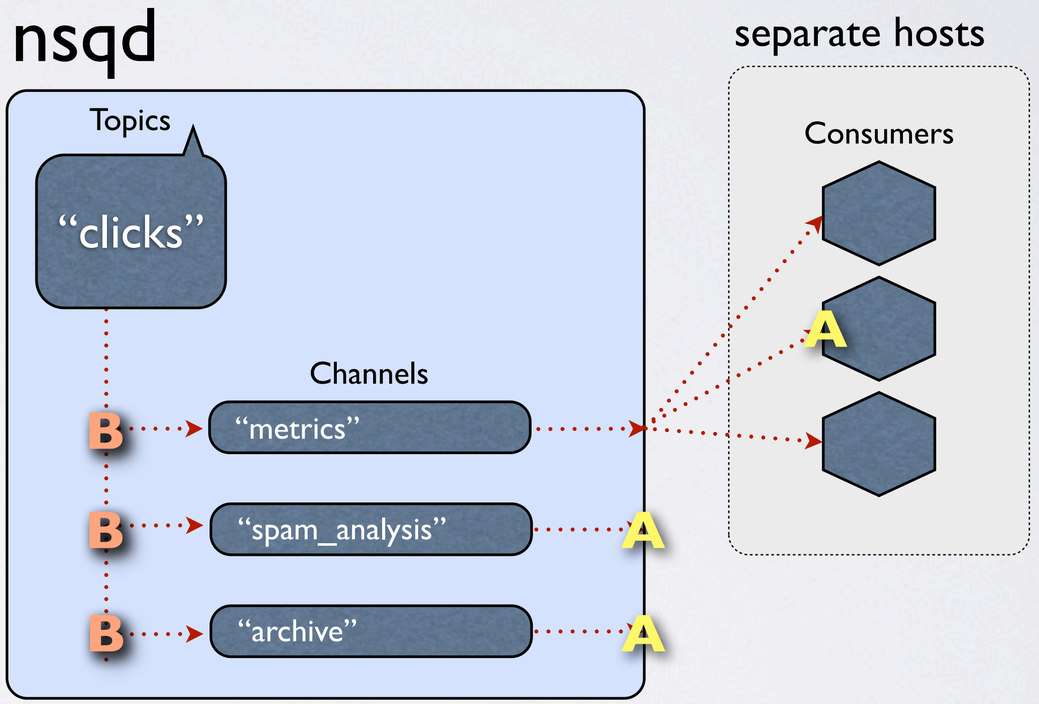
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



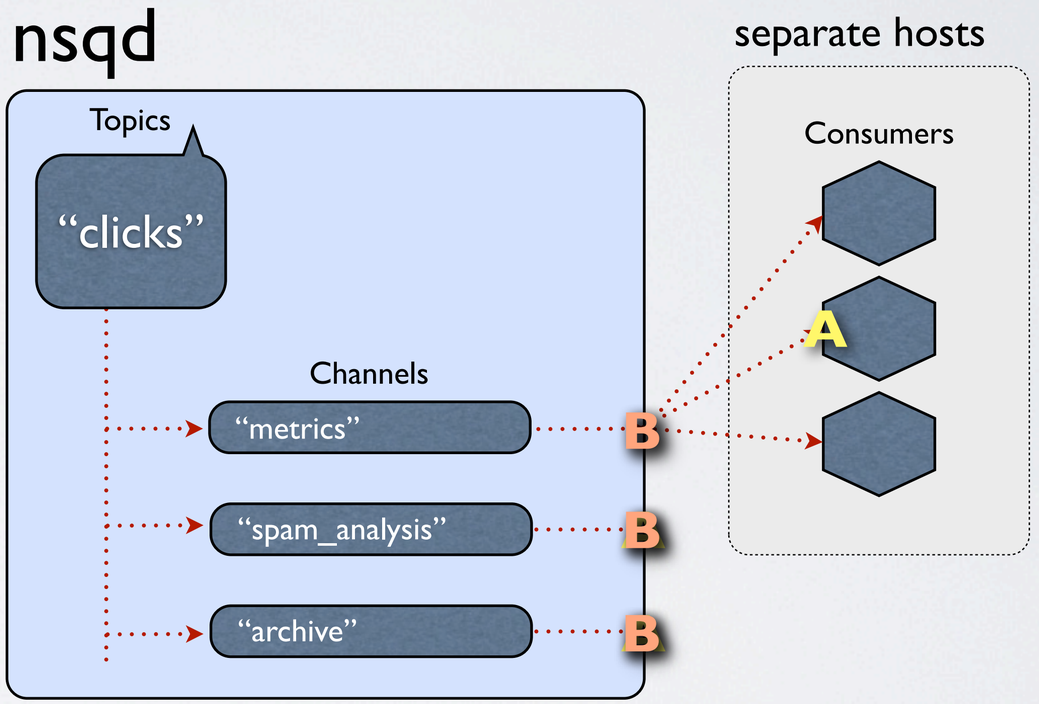
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



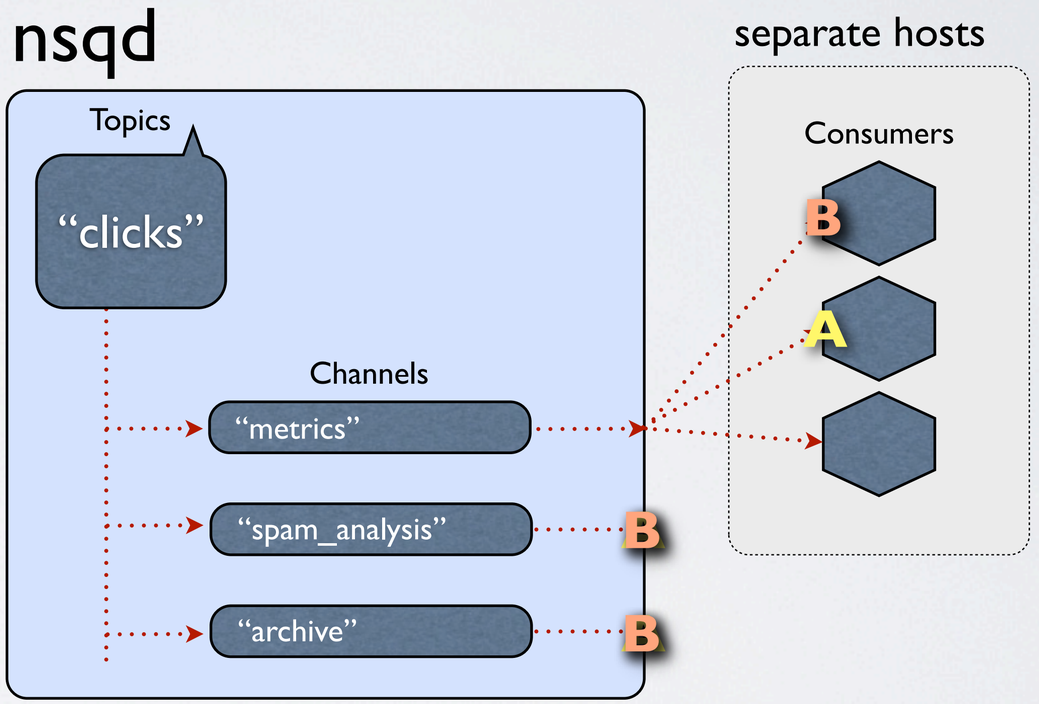
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



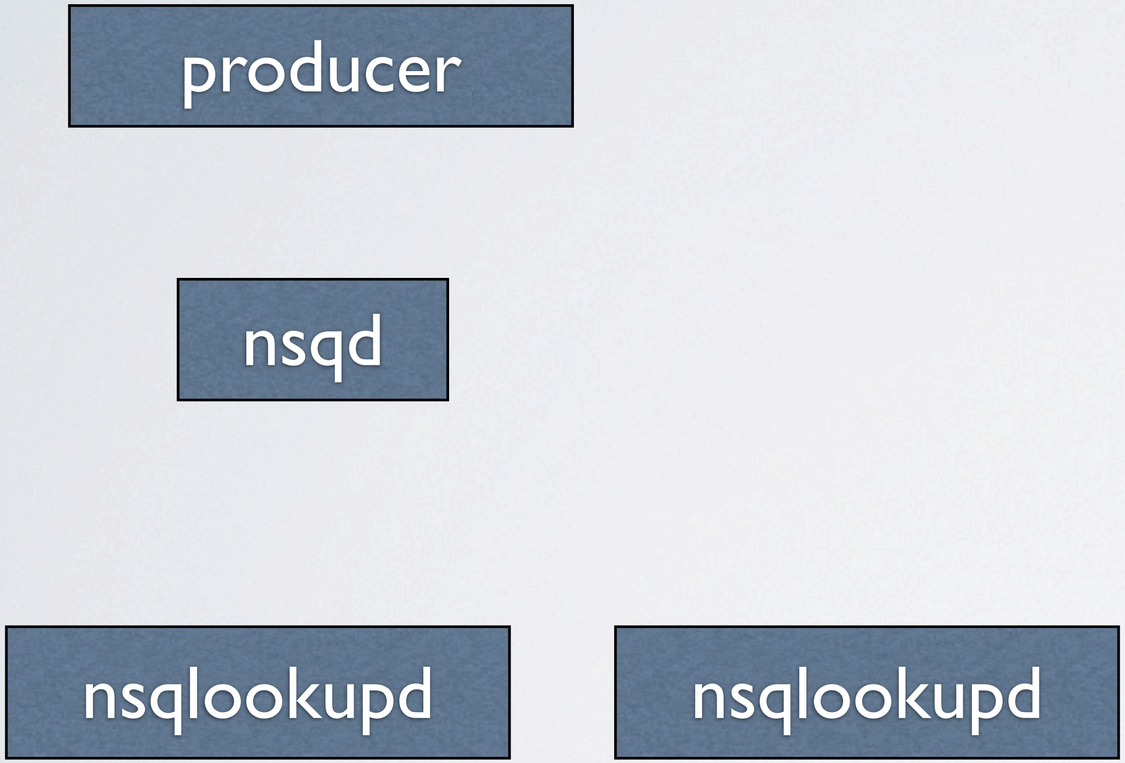
SIMPLIFY CONFIGURATION

- a **topic** is a distinct stream of messages (a single **nsqd** instance can have multiple **topics**)
- a **channel** is an independent queue for a **topic** (a **topic** can have multiple **channels**)
- consumers discover producers by querying **nsqlookupd** (a discovery service for topics)
- **topics** and **channels** are created at runtime (just start publishing/subscribing)



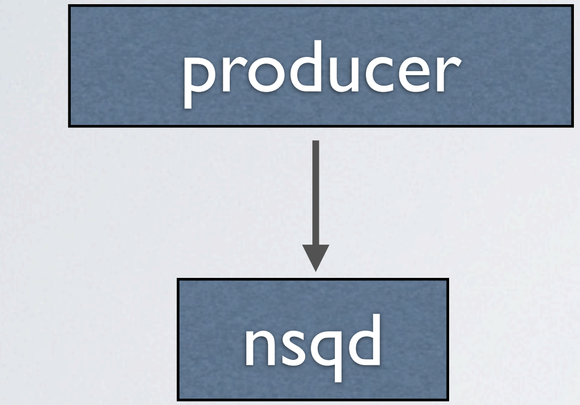
DISCOVERY

remove the need for publishers and consumers to know about each other



DISCOVERY

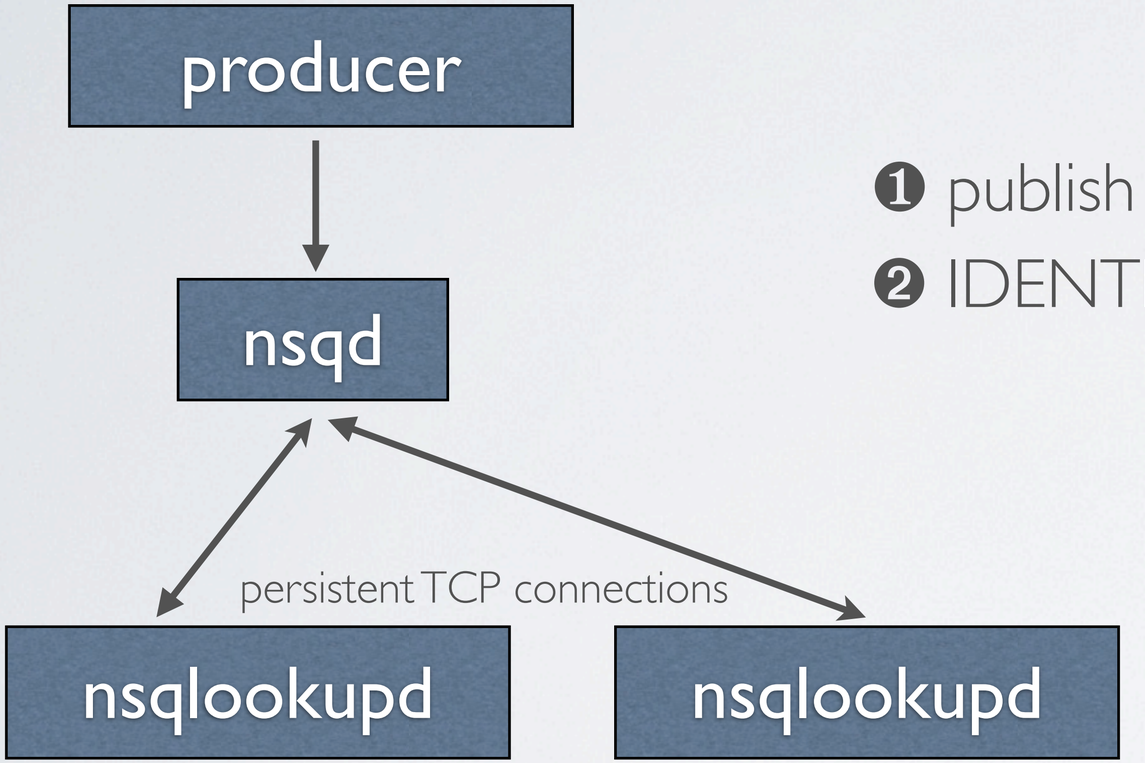
remove the need for publishers and consumers to know about each other



❶ publish msg (specifying topic)

DISCOVERY

remove the need for publishers and consumers to know about each other

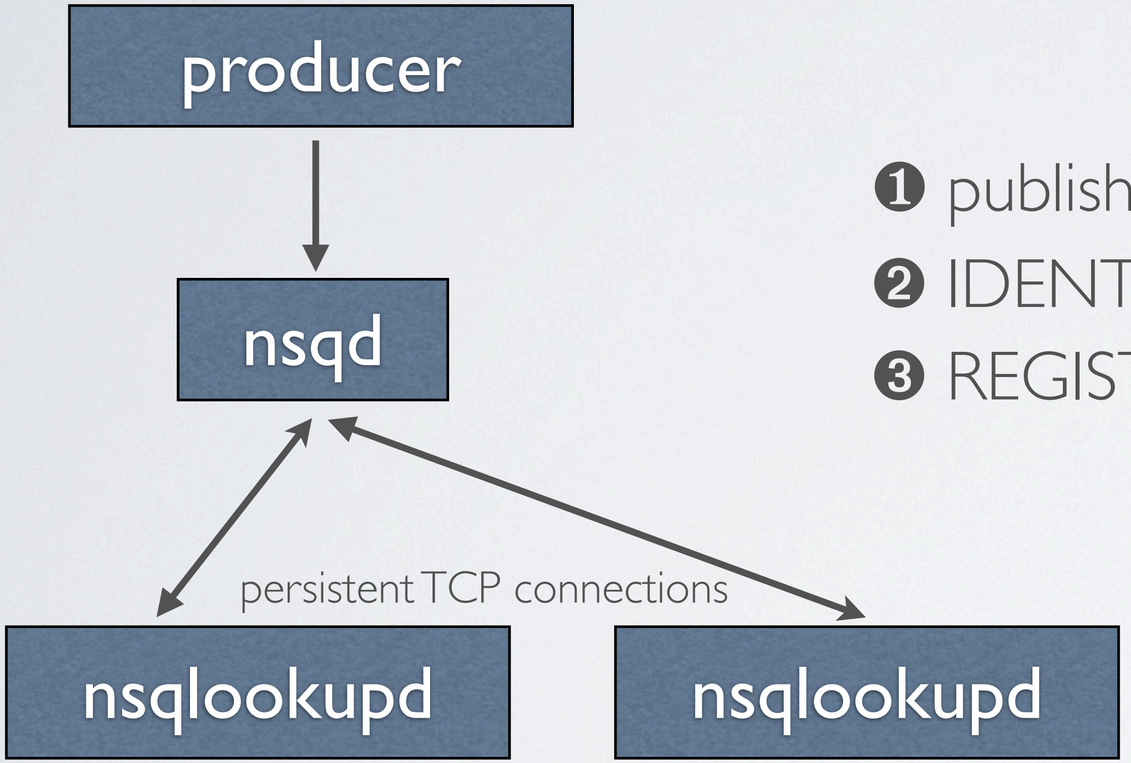


❶ publish msg (specifying topic)

❷ IDENTIFY

DISCOVERY

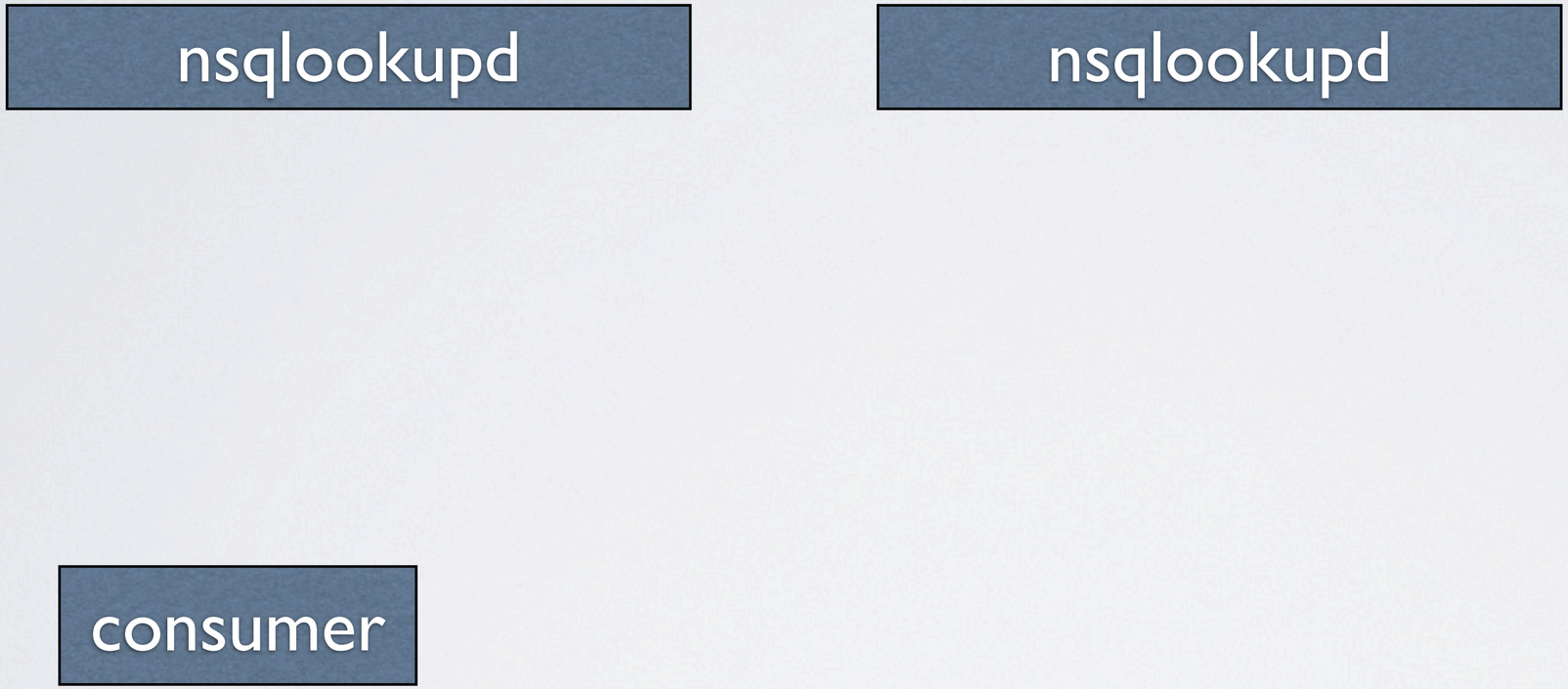
remove the need for publishers and consumers to know about each other



❶ publish msg (specifying topic)
❷ IDENTIFY
❸ REGISTER (topic/channel)

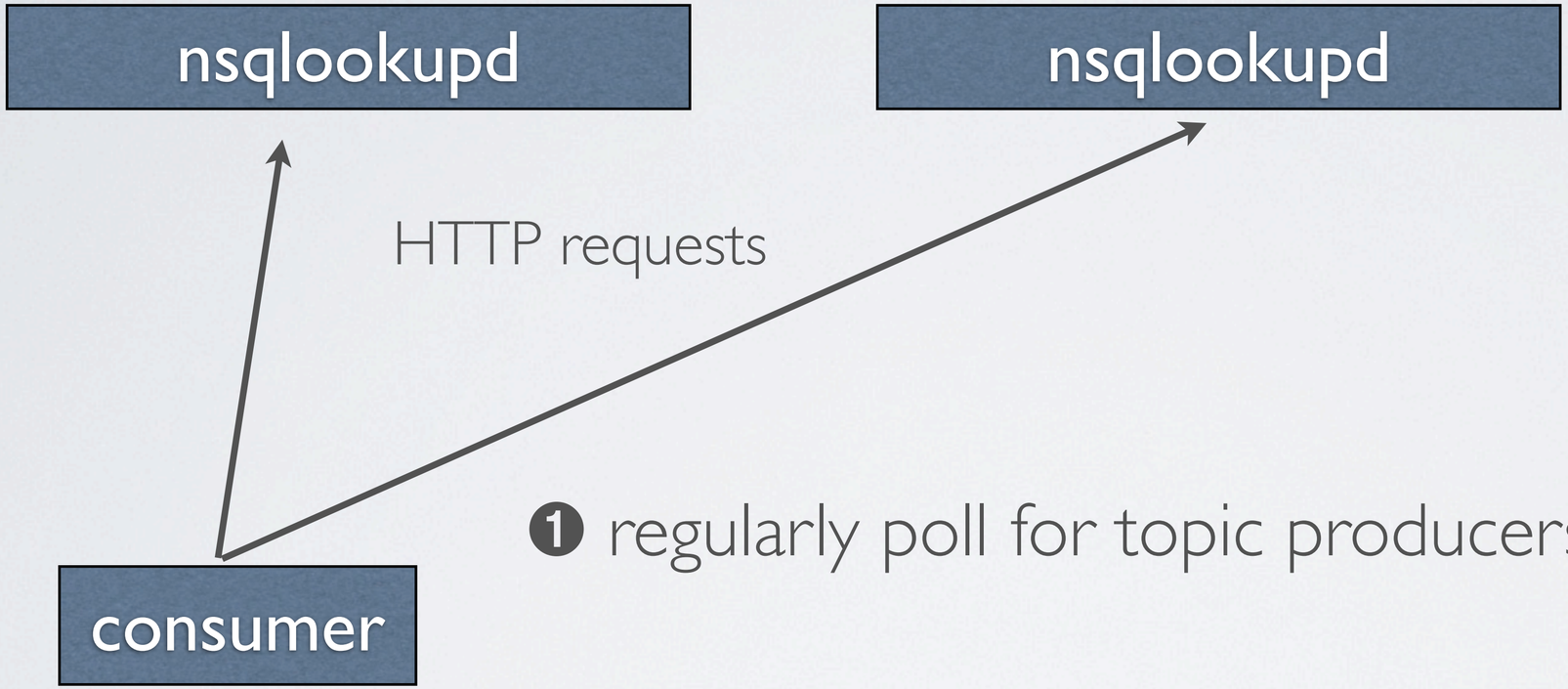
DISCOVERY (CLIENT)

remove the need for publishers and consumers to know about each other



DISCOVERY (CLIENT)

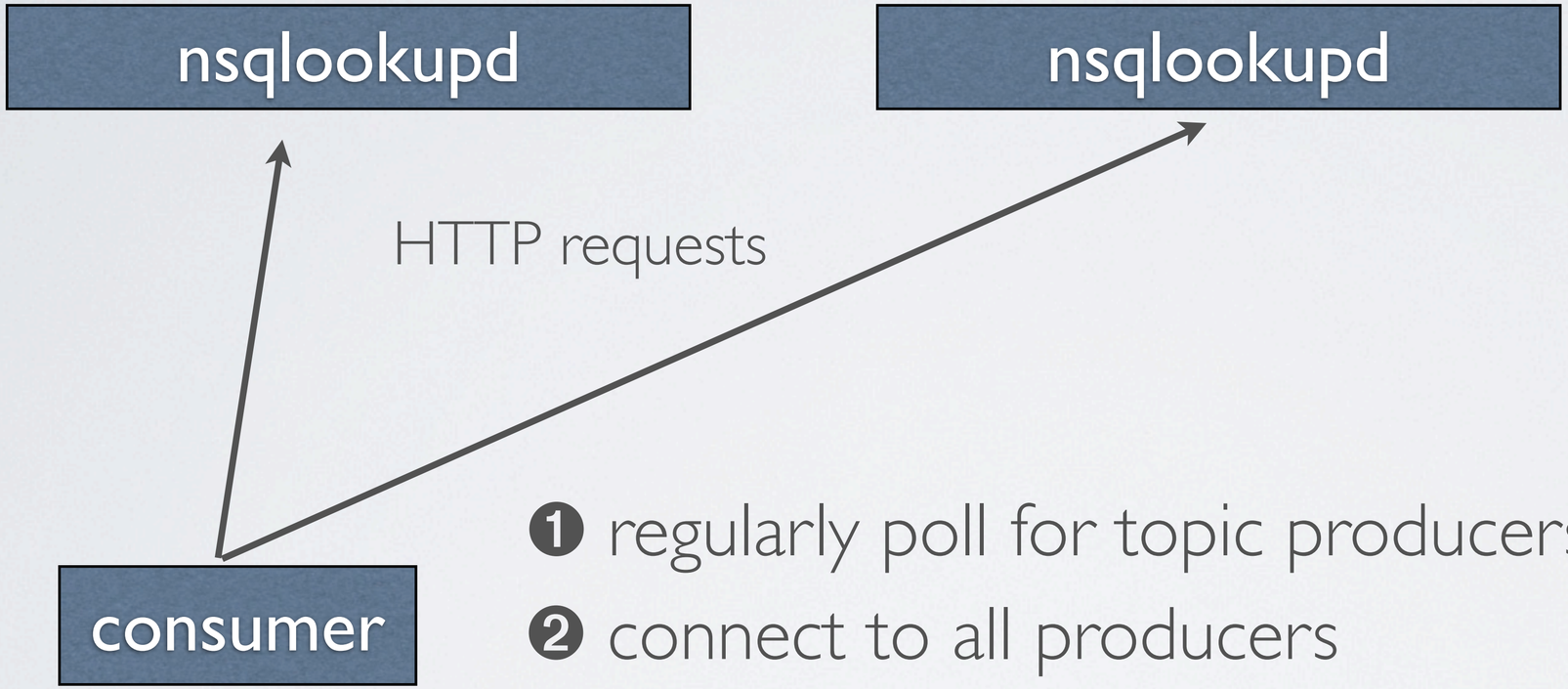
remove the need for publishers and consumers to know about each other



❶ regularly poll for topic producers

DISCOVERY (CLIENT)

remove the need for publishers and consumers to know about each other



❶ regularly poll for topic producers

❷ connect to all producers

ELIMINATE ALL THE SPOF

- easily enable *distributed* and *decentralized* topologies
- no brokers
- consumers connect to all producers
- messages are *pushed* to consumers
- **nsqlookupd** instances are *independent* and require no coordination (run a few for HA)

ELIMINATE ALL THE SPOF

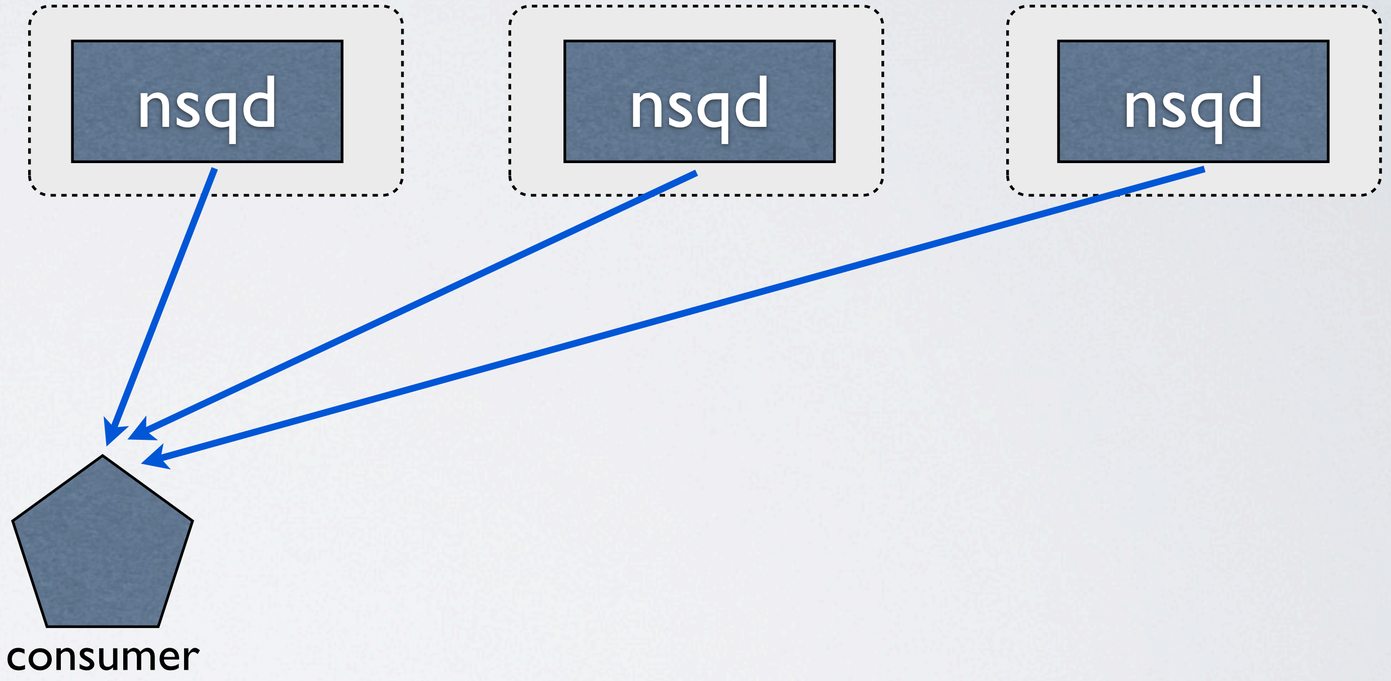
- easily enable *distributed* and *decentralized* topologies
- no brokers
- consumers connect to all producers
- messages are *pushed* to consumers
- **nsqlookupd** instances are *independent* and require no coordination (run a few for HA)

ELIMINATE ALL THE SPOF

- easily enable *distributed* and *decentralized* topologies
- no brokers
- consumers connect to all producers
- messages are *pushed* to consumers
- **nsqlookupd** instances are *independent* and require no coordination (run a few for HA)

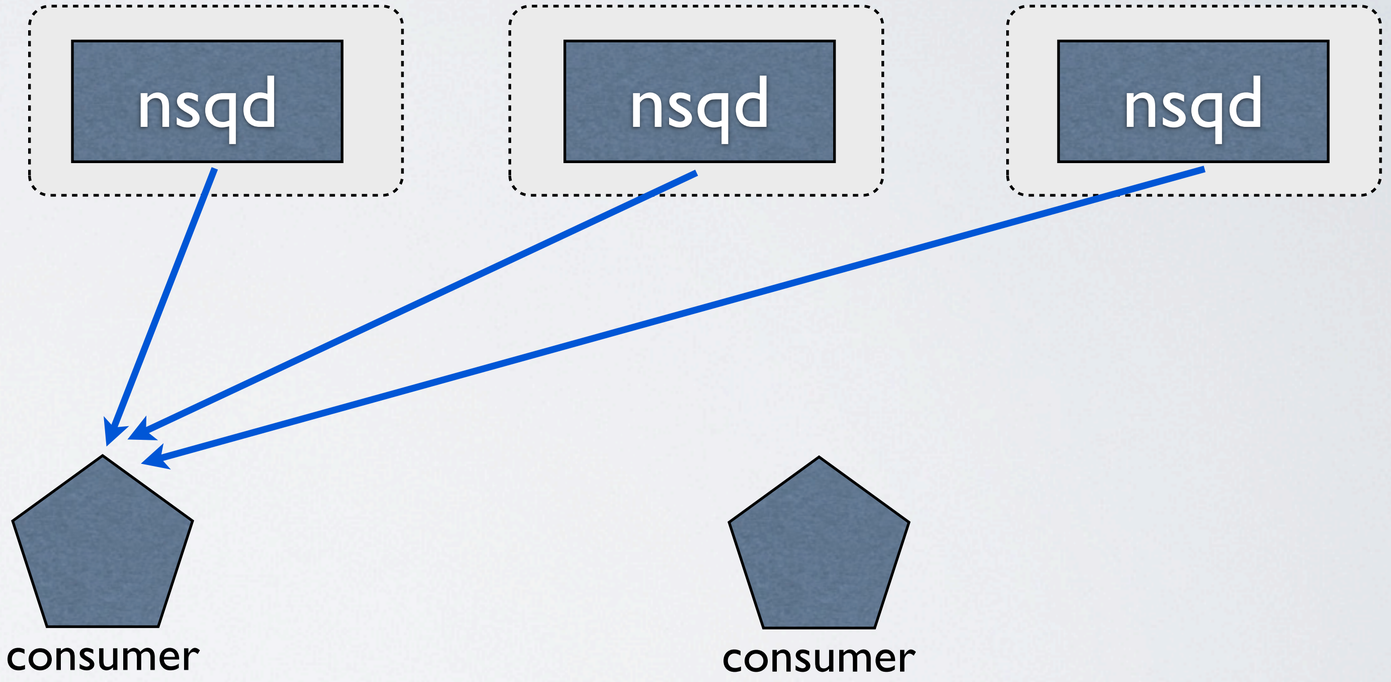
ELIMINATE ALL THE SPOF

- easily enable *distributed* and *decentralized* topologies
- no brokers
- consumers connect to all producers
- messages are *pushed* to consumers
- **nsqlookupd** instances are *independent* and require no coordination (run a few for HA)



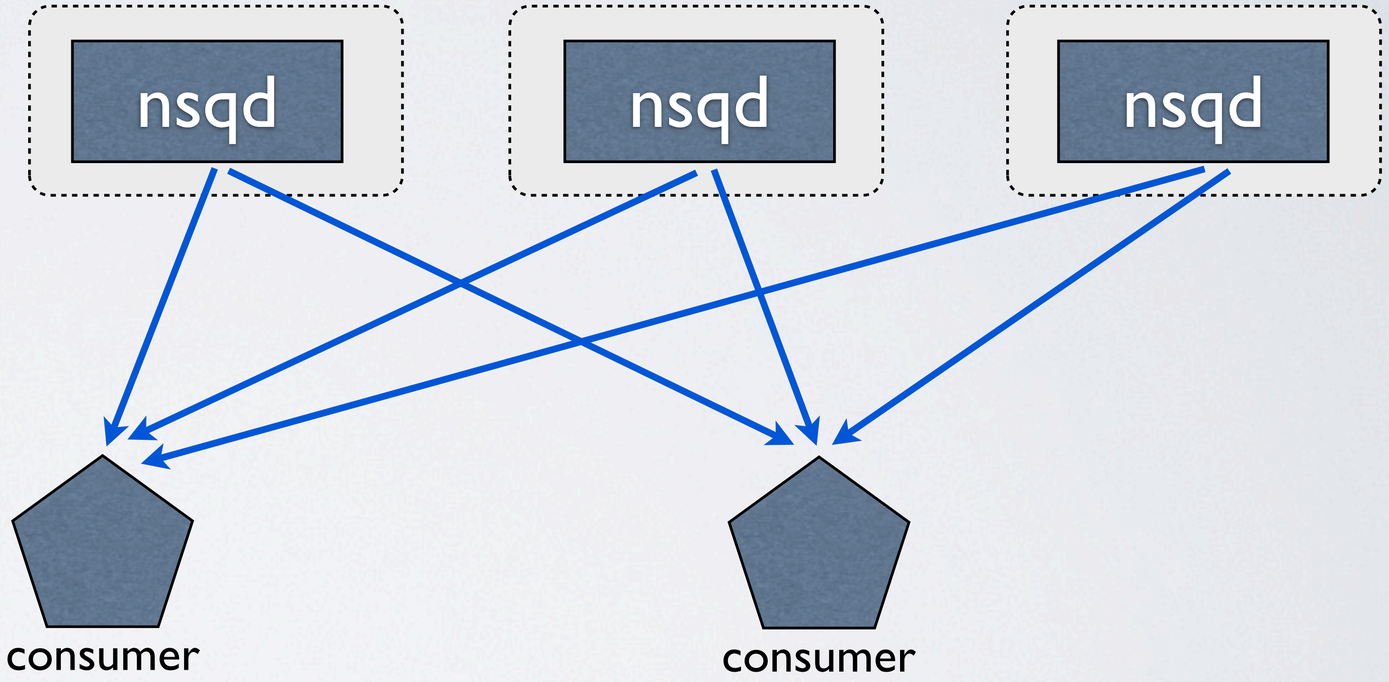
ELIMINATE ALL THE SPOF

- easily enable *distributed* and *decentralized* topologies
- no brokers
- consumers connect to all producers
- messages are *pushed* to consumers
- **nsqlookupd** instances are *independent* and require no coordination (run a few for HA)



ELIMINATE ALL THE SPOF

- easily enable *distributed* and *decentralized* topologies
- no brokers
- consumers connect to all producers
- messages are *pushed* to consumers
- **nsqlookupd** instances are *independent* and require no coordination (run a few for HA)



MESSAGE GUARANTEES

- messages are delivered *at least* once
- *handling* is guaranteed by the protocol:
  - nsqd sends a message and stores it temporarily
  - client replies FIN (finish) or REQ (re-queue)
  - if client does not reply message is *automatically* re-queued
- any single *nsqd* instance failure can result in message loss (can be mitigated)

EFFICIENCY

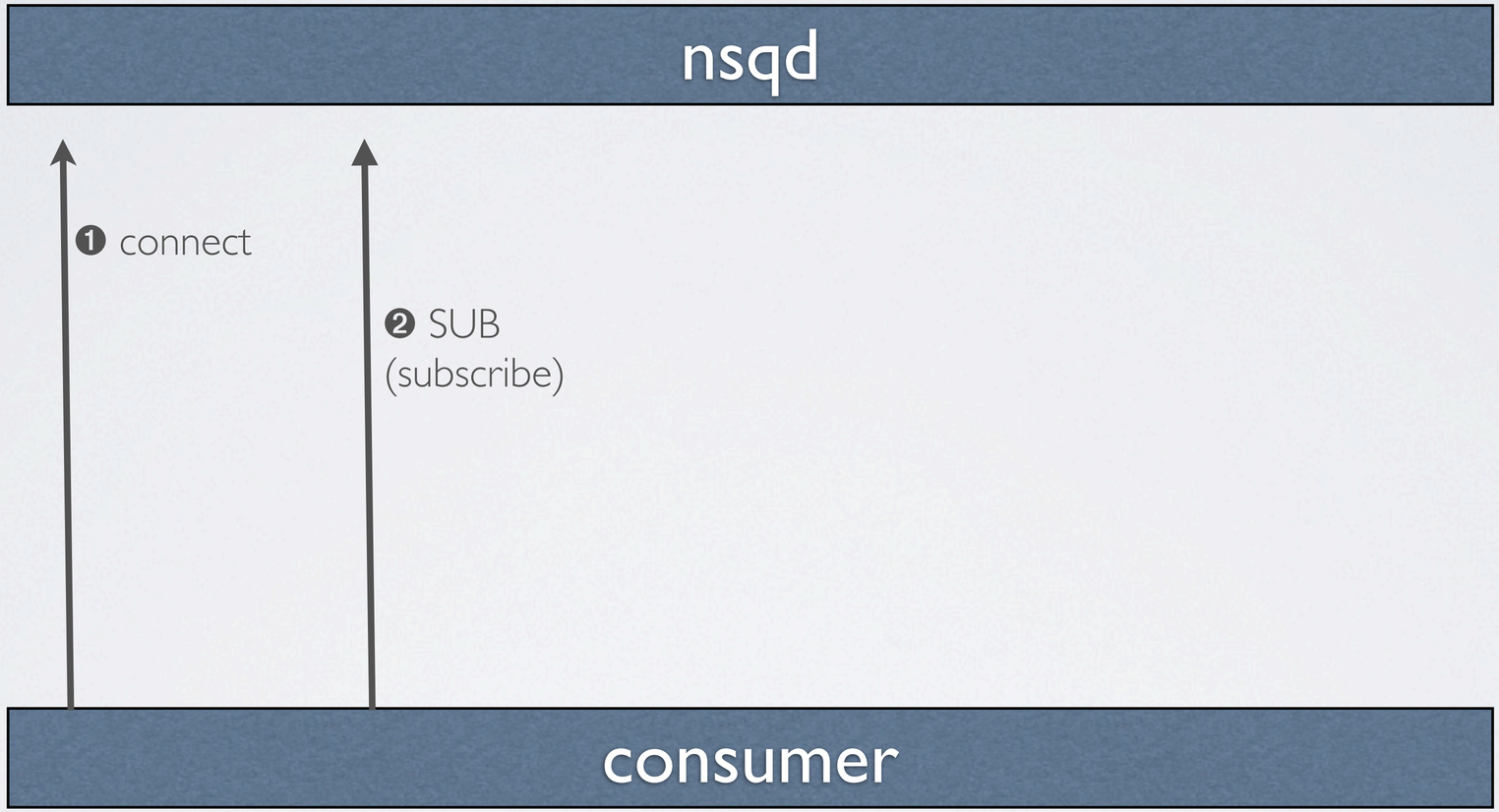
get RDY

EFFICIENCY

get RDY

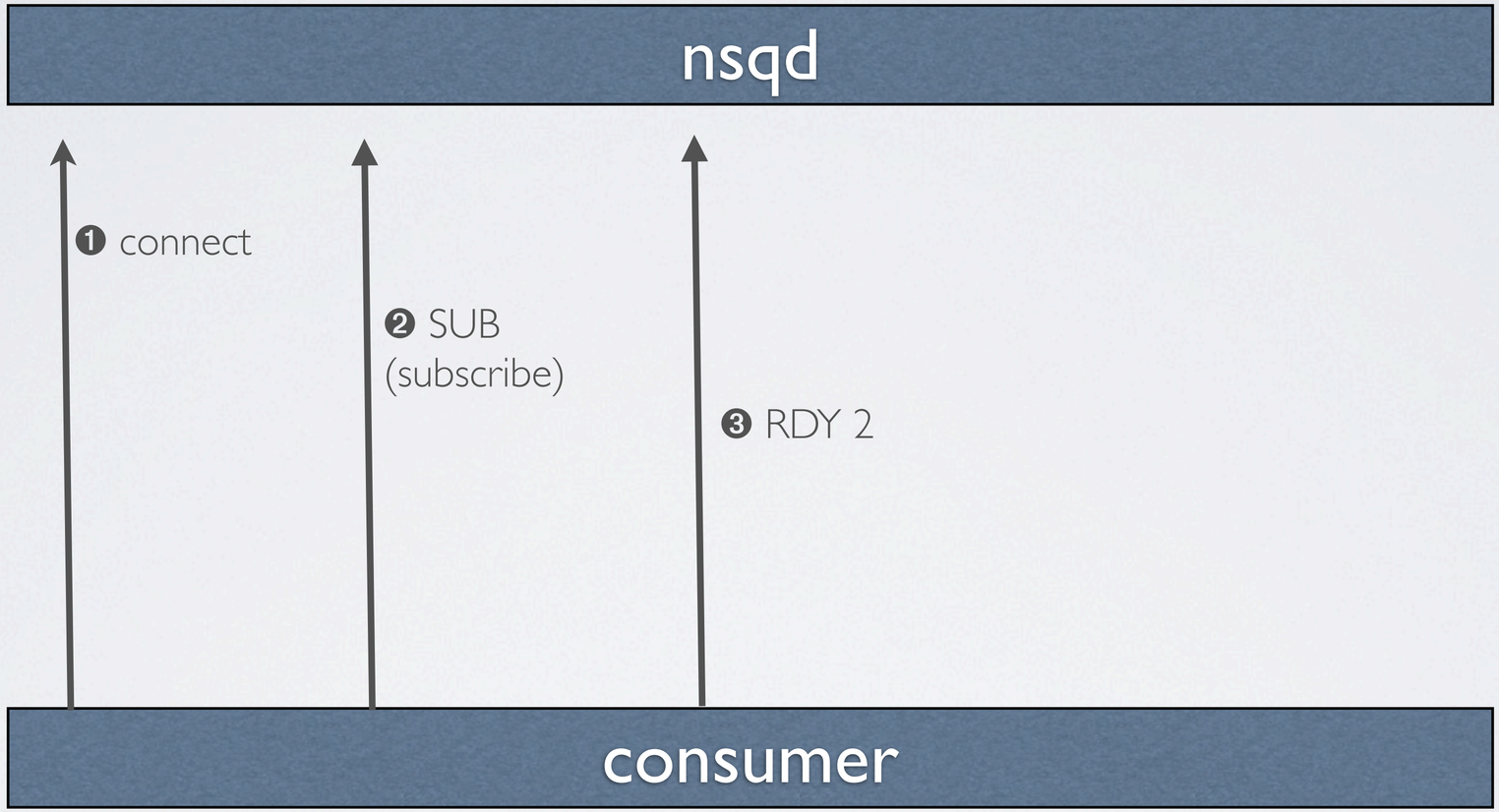
EFFICIENCY

get RDY



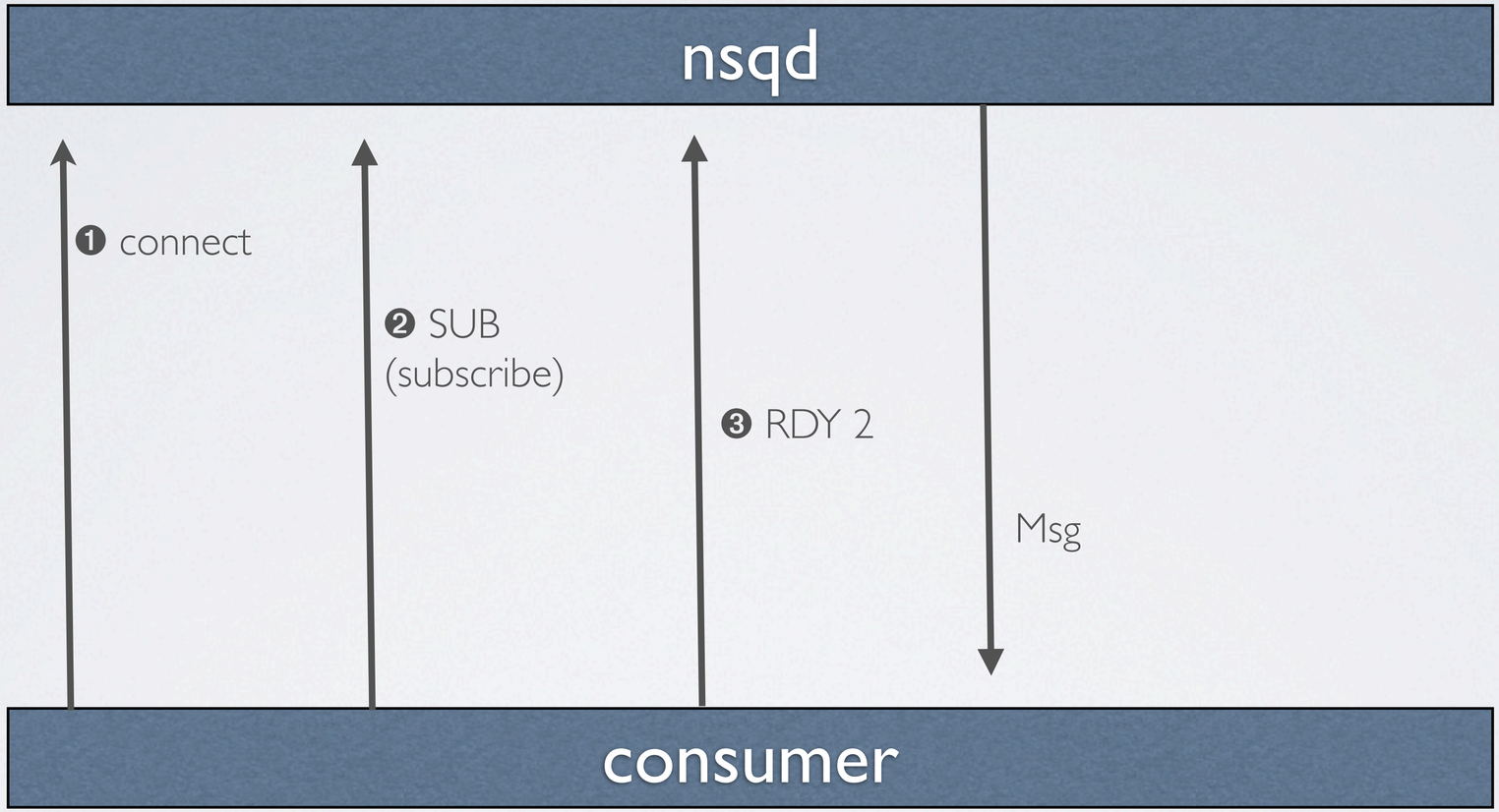
EFFICIENCY

get RDY



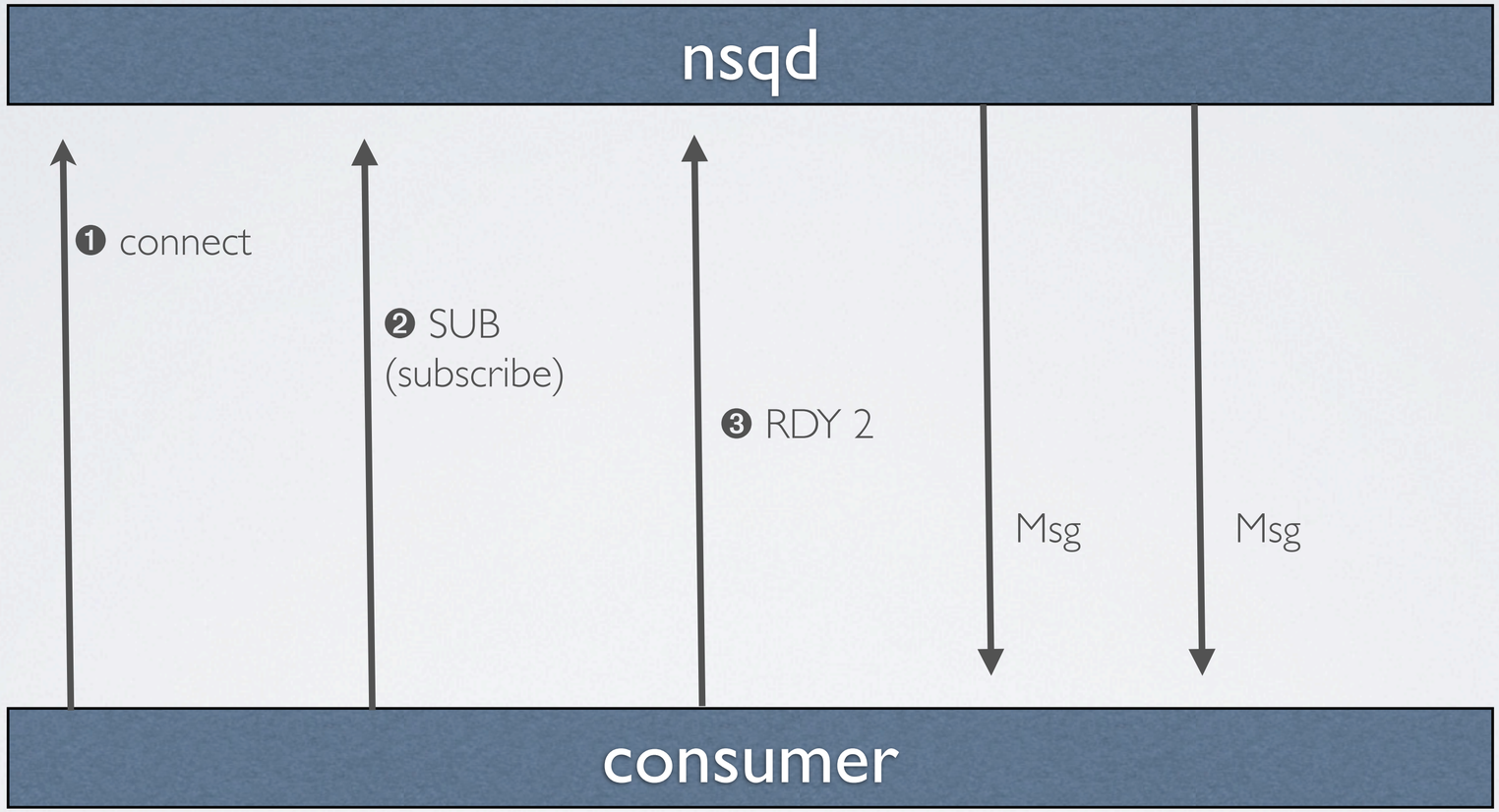
EFFICIENCY

get RDY



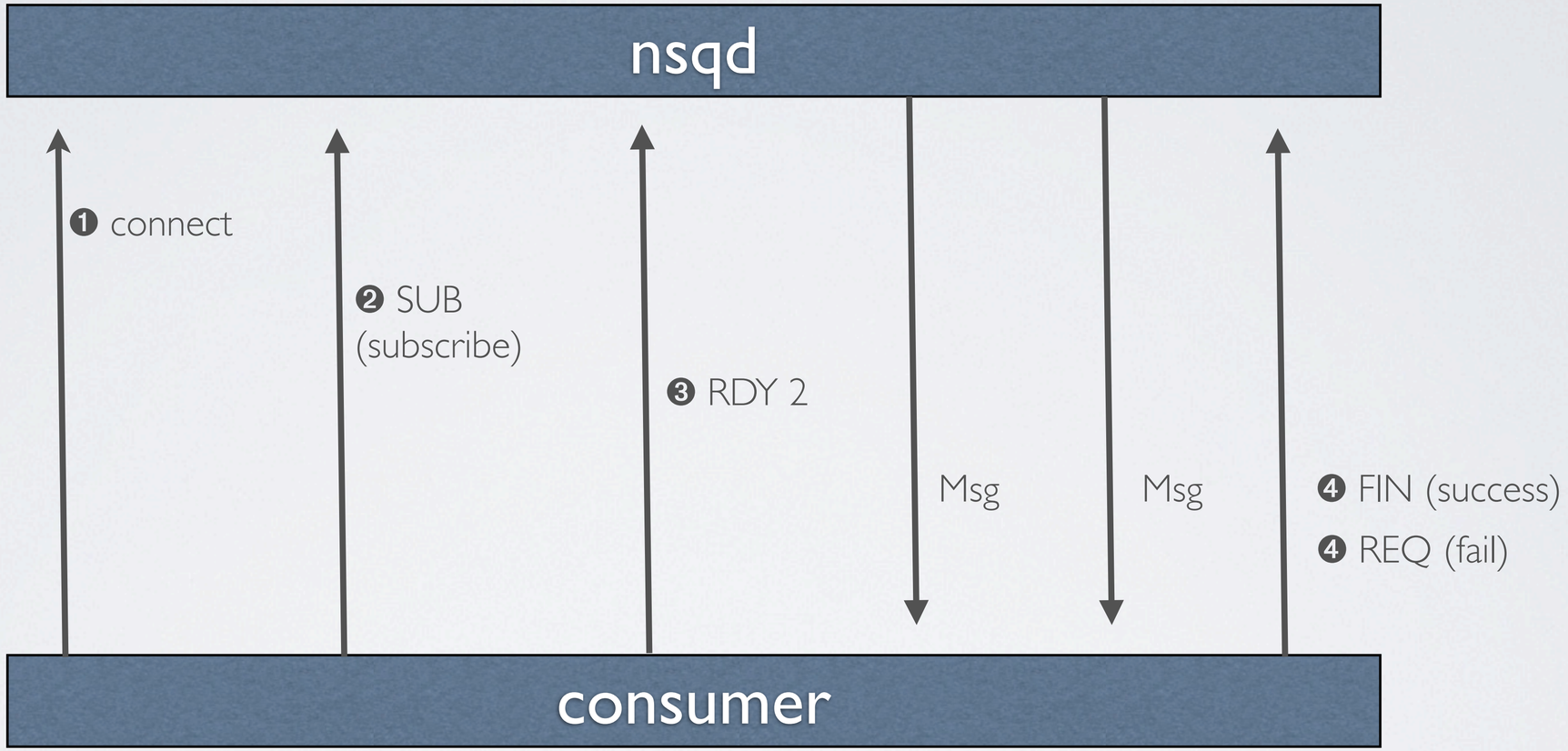
EFFICIENCY

get RDY



EFFICIENCY

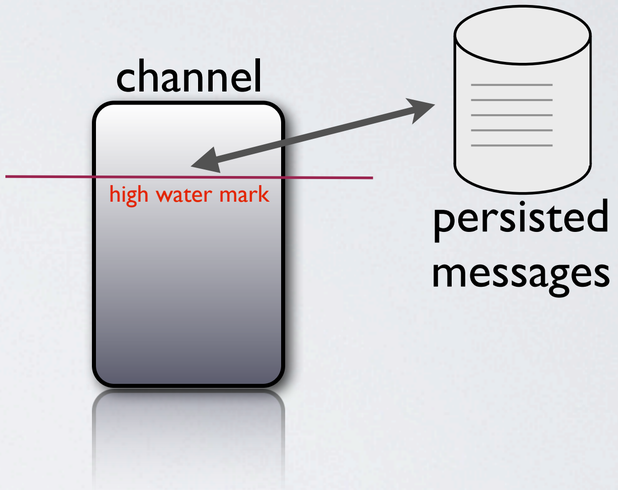
get RDY



QUEUES

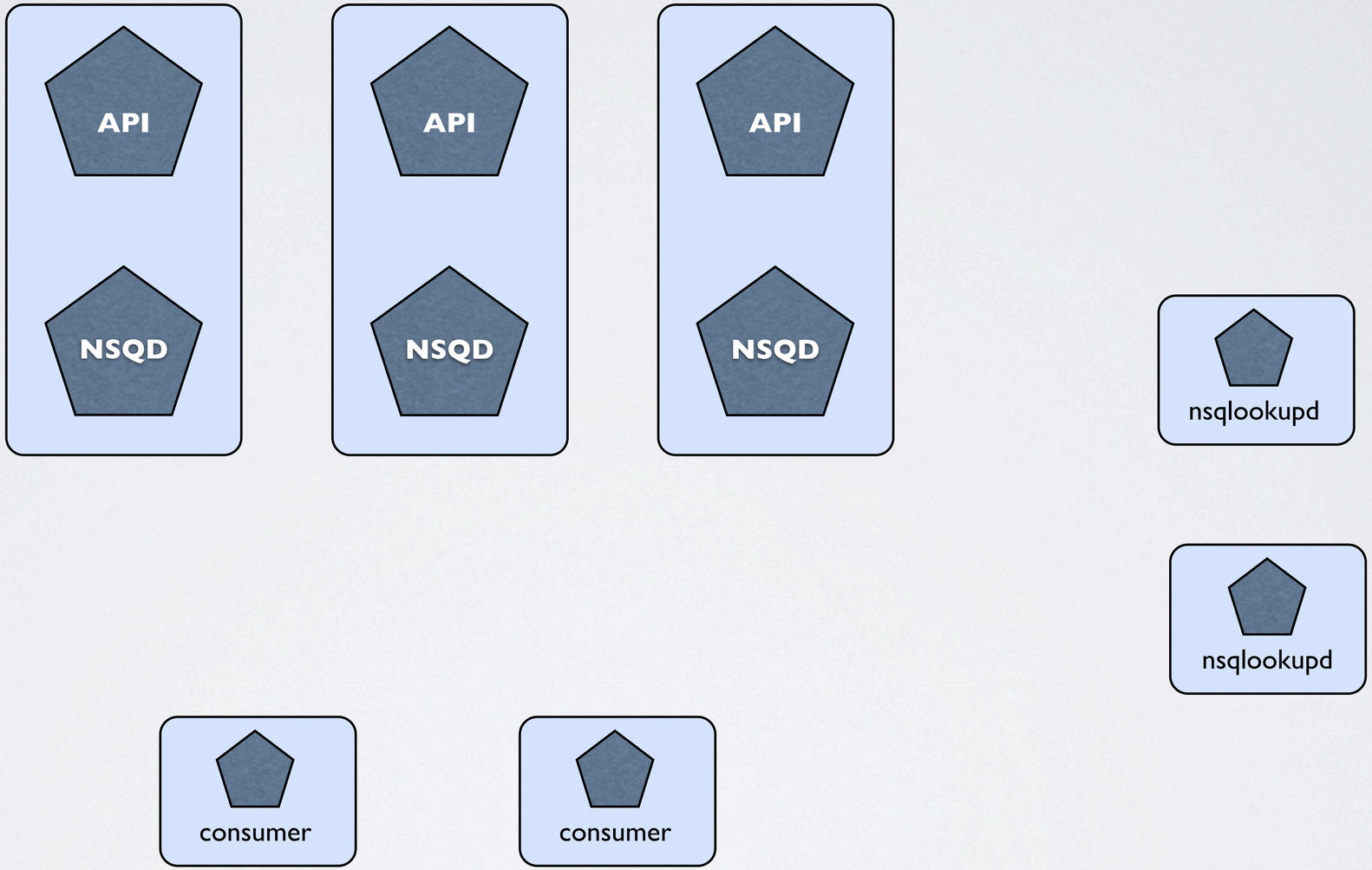
buffer *this*

- **topics** and **channels** are *independent* queues
- queues have arbitrary high water marks (after which messages transparently read/write to disk, bounding memory footprint)
- supports channel-independent degradation and recovery
- 10 lines of Go

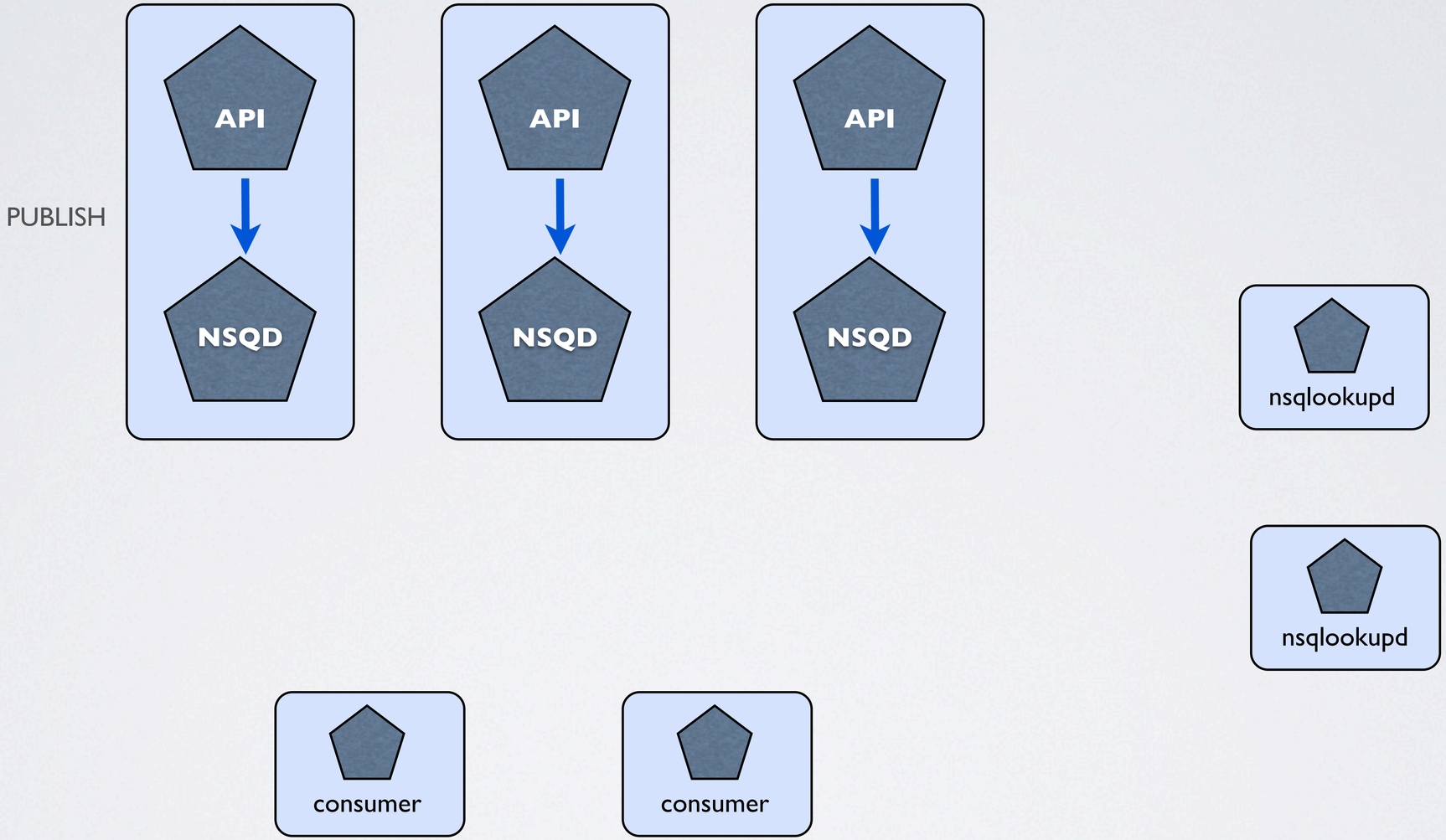



```
for msg := range c.incomingMsgChan {¬
    select {¬
    case c.memoryMsgChan <- msg:¬
    default:¬
        err := WriteMessageToBackend(&msgBuf, msg, c)¬
        if err != nil {¬
            // log whatever¬
        }¬
    }¬
}
```

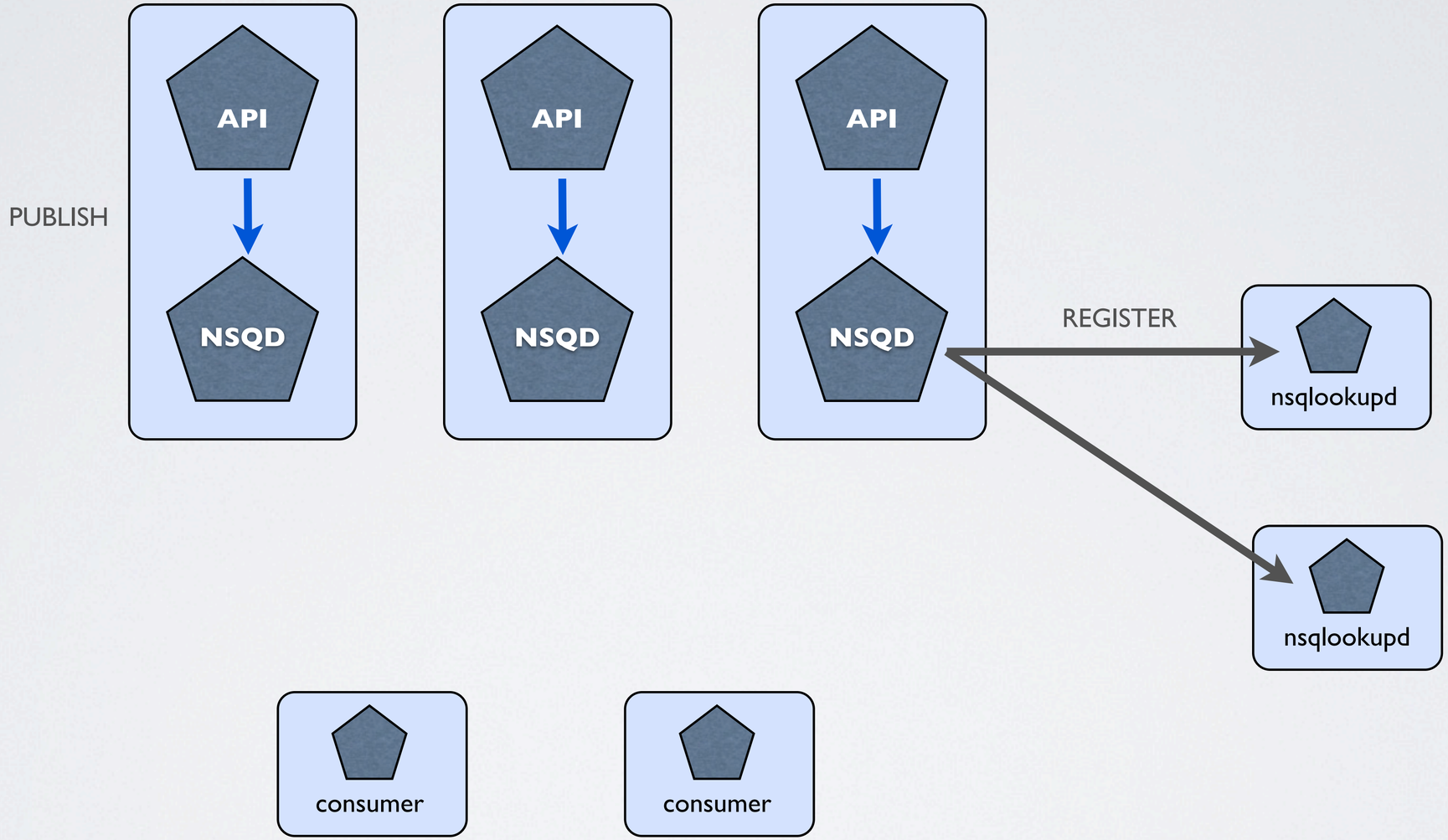
NSQ ARCHITECTURE



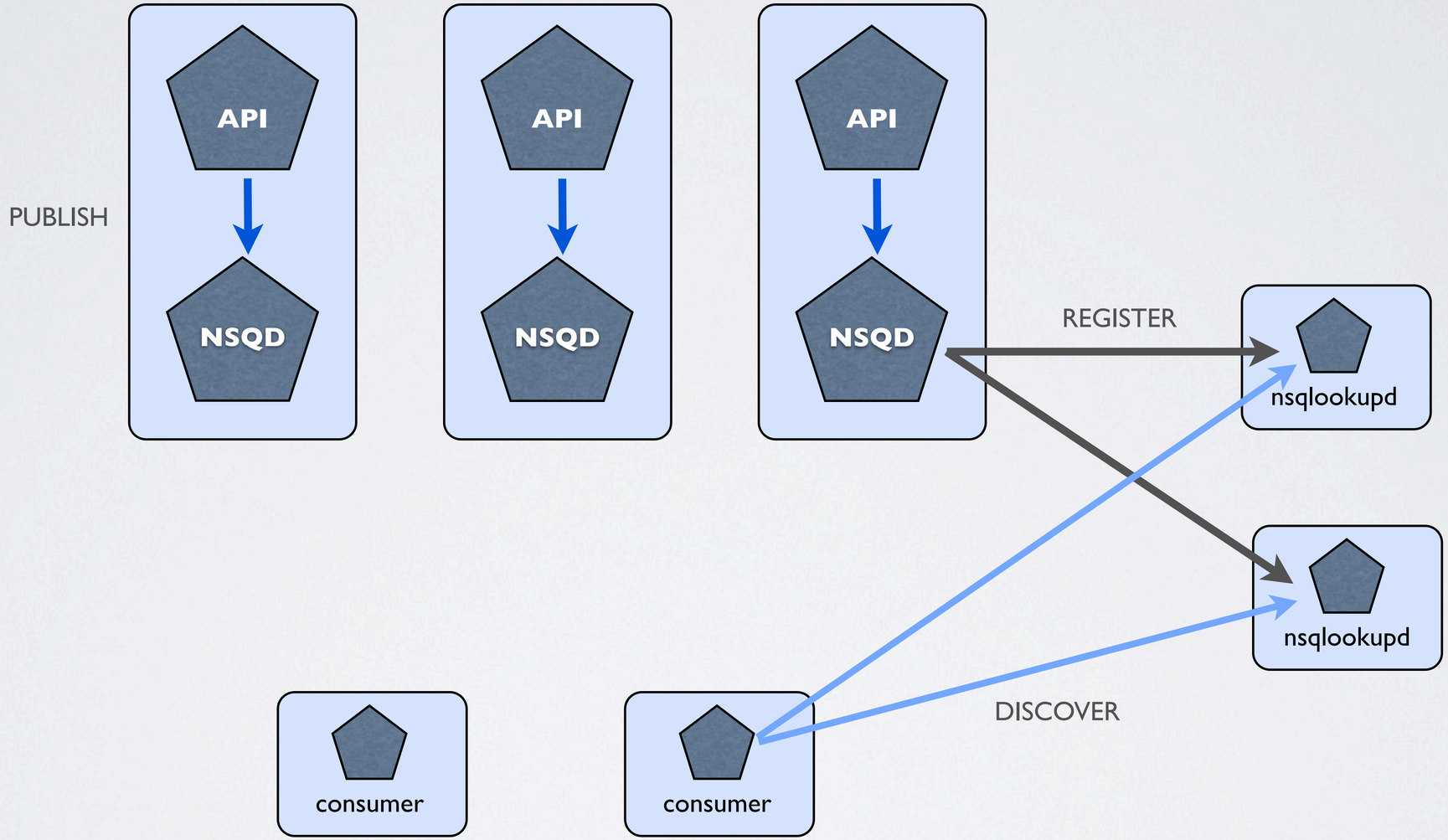
NSQ ARCHITECTURE



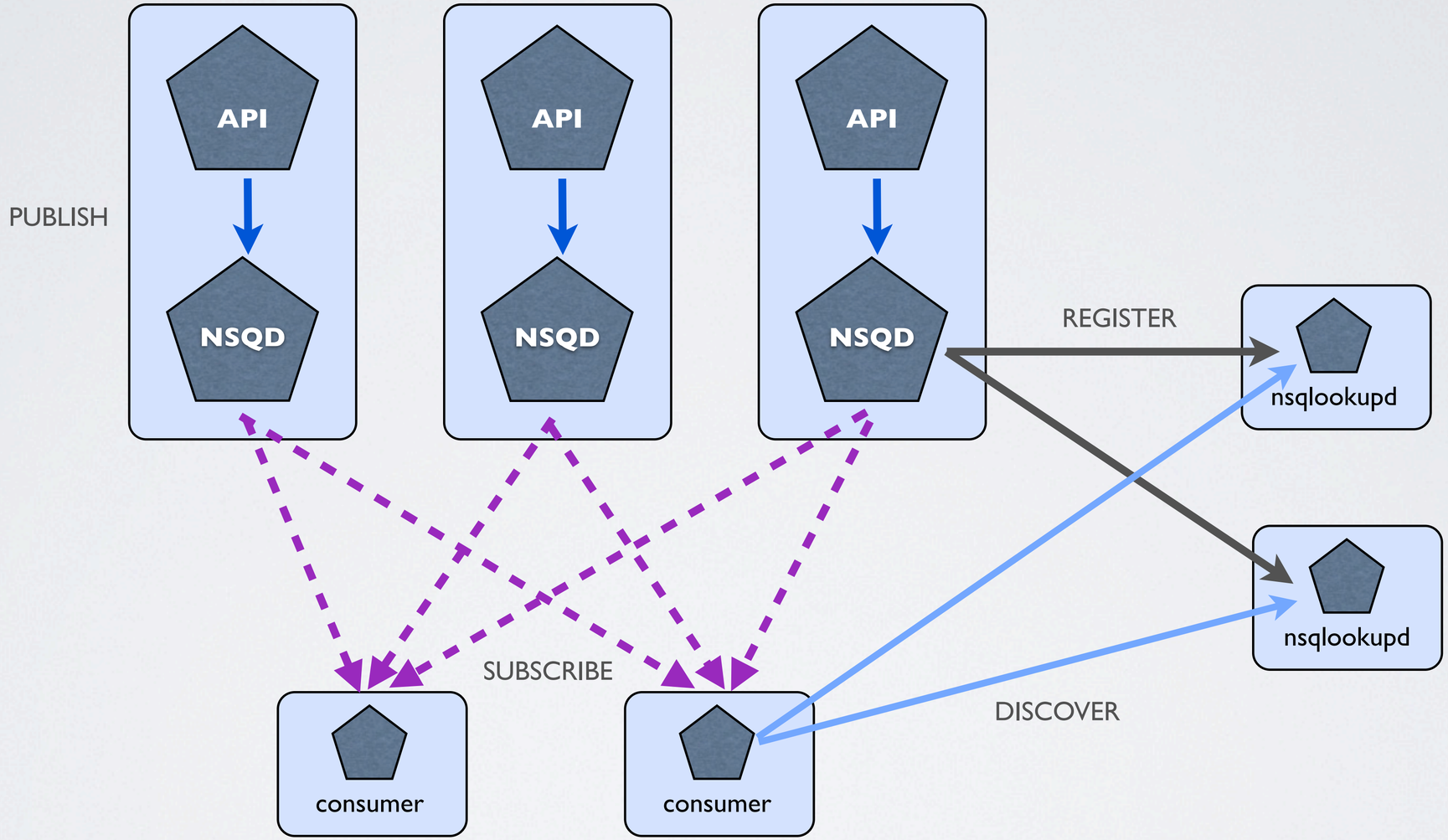
NSQ ARCHITECTURE



NSQ ARCHITECTURE

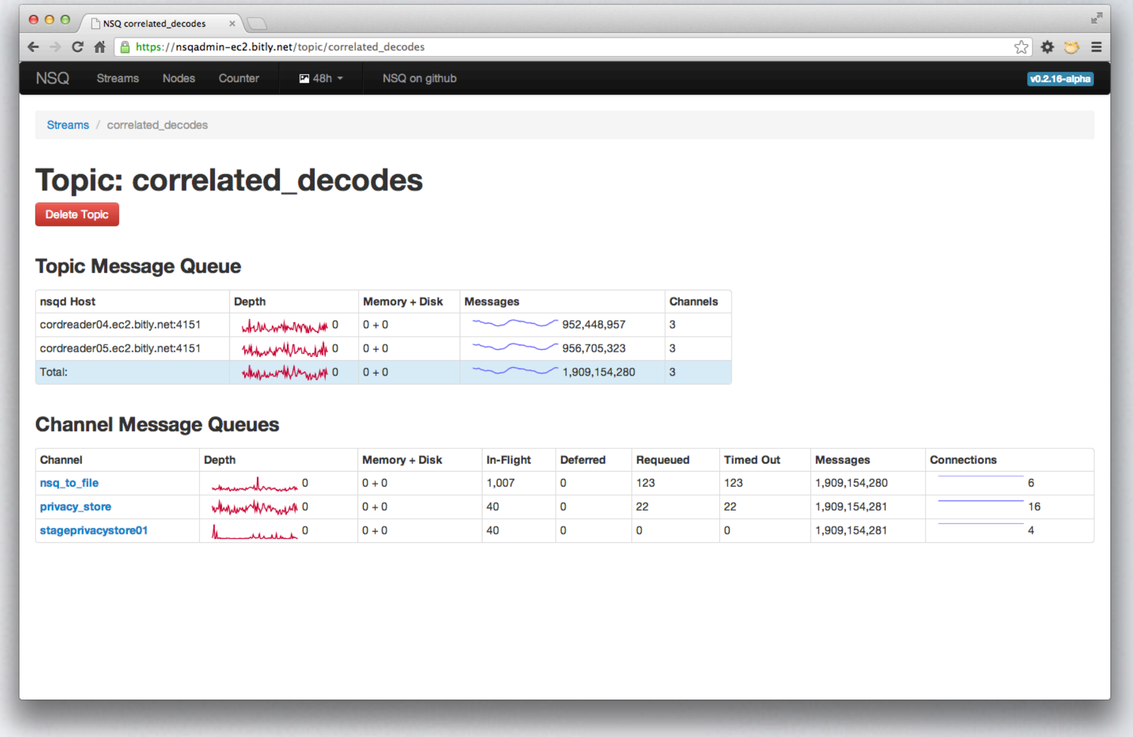


NSQ ARCHITECTURE



TOOLING

- **nsqadmin** provides a web interface to administrate and introspect an NSQ cluster at runtime
  - empty, pause, delete channels
- **nsq_to_http** - utility that helps transport an aggregate stream over HTTP
- **nsq_to_file** - utility that safely persists an aggregated stream to disk



ONE MORE THING

- **#ephemeral** channels - runtime introspection
  - no backup beyond channel high water mark
  - automatically go away when last client disconnects
- server side channel *pausing*
  - administratively stop the flow of messages from a channel to its clients
  - no message loss (queue backs up)
  - really $#%^ing awesome for operations

IMPLEMENTATION

GO LESSONS LEARNED

- don't be afraid of the **sync** package
- goroutines are ***cheap*** not ***free***
- watch your allocations (string() is costly, re-use buffers)
- use anonymous structs for arbitrary JSON
- no built-in per-request HTTP timeouts
- synchronizing goroutine exit is hard - log each cleanup step in long-running goroutines
- *select* skips nil channels

```
for {
    if canRead() {
        dataRead = d.readOne()
        r = d.readChan
    } else {
        r = nil
    }

    select {
    case r <- dataRead:
    case dataWrite := <-d.writeChan:
        d.writeOne(dataWrite)
    case <-d.exitChan:
        goto exit
    }
}
```

IMPORT PATH

- our git workflow involves *hardcore forking action*
- re-writing import paths sucks
- $ go tool install_as --import-path=github.com/bitly/nsq/nsq
  - https://github.com/mreiferson/go-install-as
- also - relative imports are OK

CLIENTS

- Go Client - https://gist.github.com/4039222
- Synchronous Python Client - https://gist.github.com/3925081
- Async Python Client - https://gist.github.com/3925092

DEMO

Thanks

@imsnakes & **@jehiah**

https://github.com/bitly/nsq

shoutouts to **@danielhfrank**, **@ploxiln**, and **@mccutchen**