THE PATTERNS OF DISTRIBUTED LOGGING AND CONTAINERS

CloudNativeCon Europe 2017
March 30, 2017

Satoshi Tagomori (@tagomoris)
Treasure Data, Inc.

SATOSHI TAGOMORI

(@tagomoris)

Fluentd, MessagePack-Ruby, Norikra, ...

Treasure Data, Inc.

TREASURE
DATA

fluentd

CLOUD NATIVE
COMPUTING FOUNDATION

1. Microservices, Containers and Logging
2. Scaling Logging Platform
3. Patterns: Source/Destination -side Aggregation
4. Patterns: Scaling Up/Out Destination
5. Practices

MICROSERVICES, CONTAINERS AND LOGGING

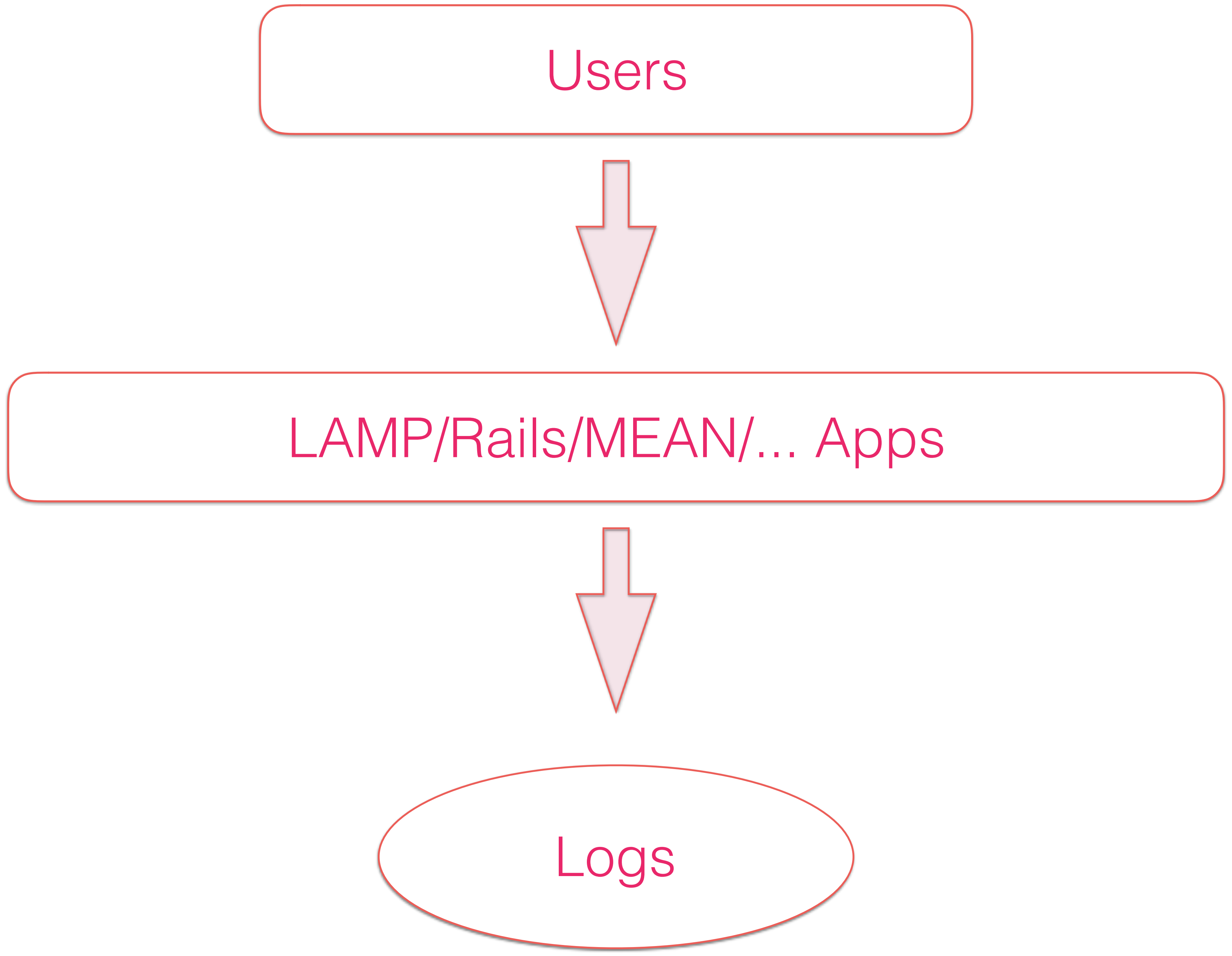
Logging in Industries

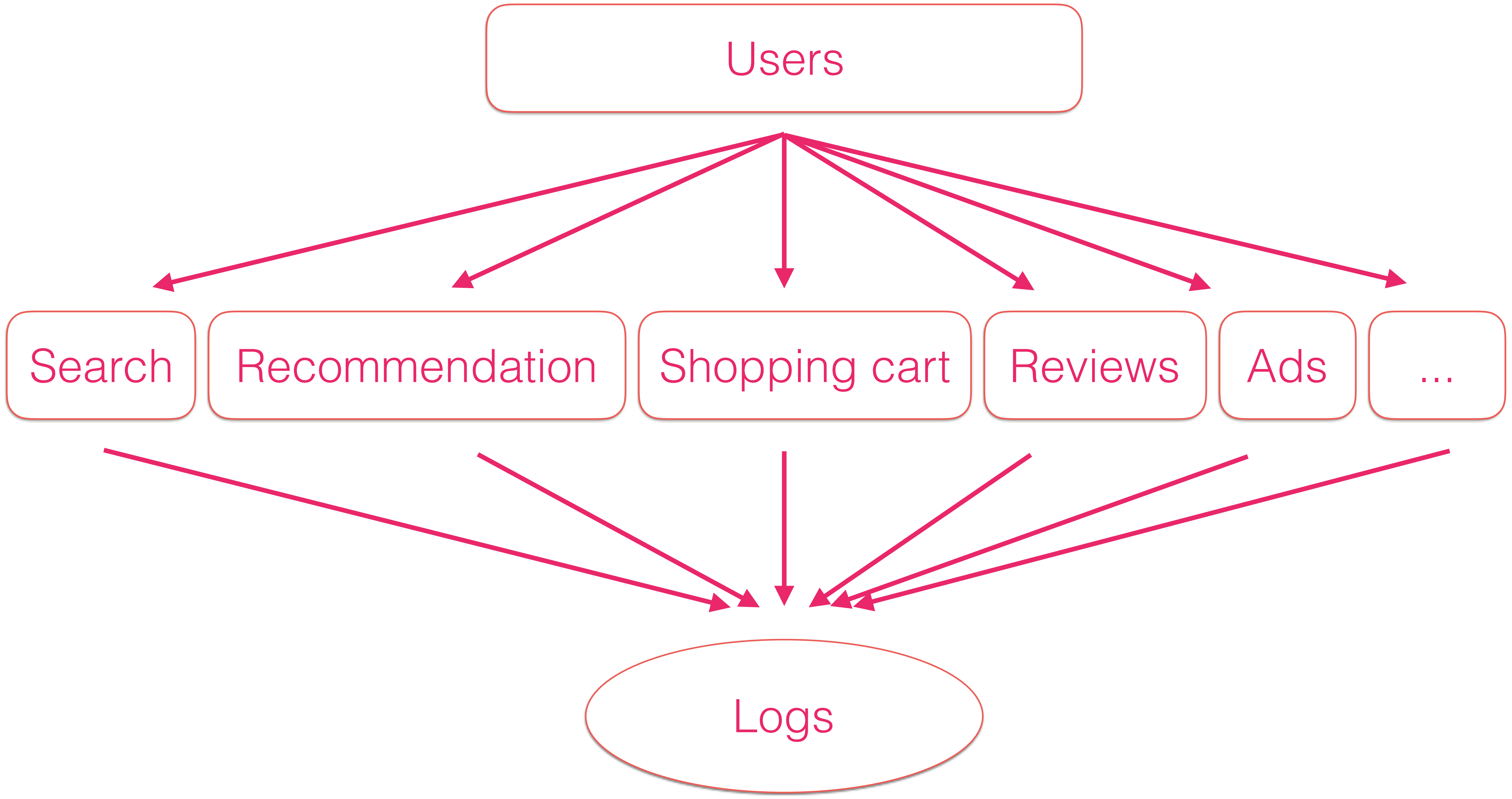
- Service Logs
  - Web access logs
  - Ad logs
  - Commercial transaction logs for analytics (EC, Game, ...)
- System Logs
  - Syslog and other OS logs
  - Audit logs
  - Performance metrics

Logs for Growth

Logs for Stability

Microservices and Logging





Monolithic service

Microservices

Microservices and Containers

- Microservices
  - Isolated dependencies
  - Agile deployment
- Containers
  - Isolated environments & resources
  - Simple pull&restart deployment
  - Less overhead, high density

Logging Challenges with Microservices/Containers

- Containerization changes everything:
  - No permanent storages
  - No fixed physical/network addresses
  - No fixed mapping between servers and roles

Logging Challenges with Microservices/Containers

- Containerization changes everything:
  - No permanent storages
  - No fixed physical/network addresses
  - No fixed mapping between servers and roles

Transfer Logs to Anywhere ASAP

Logging Challenges with Microservices/Containers

- Containerization changes everything:
  - No permanent storages
  - No fixed physical/network addresses
  - No fixed mapping between servers and roles

Push Logs From Containers

Logging Challenges with Microservices/Containers

- Containerization changes everything:
  - No permanent storages
  - No fixed physical/network addresses
  - No fixed mapping between servers and roles

Label Logs With Service Names/Tags

Logging Challenges with Microservices/Containers

- Containerization changes everything:
  - No permanent storages
  - No fixed physical/network addresses
  - No fixed mapping between servers and roles

Label Logs With Service Names/Tags

Parse Logs & Label Values At Source

Structured Logs

Structured Logs: tag, time, key-value pairs

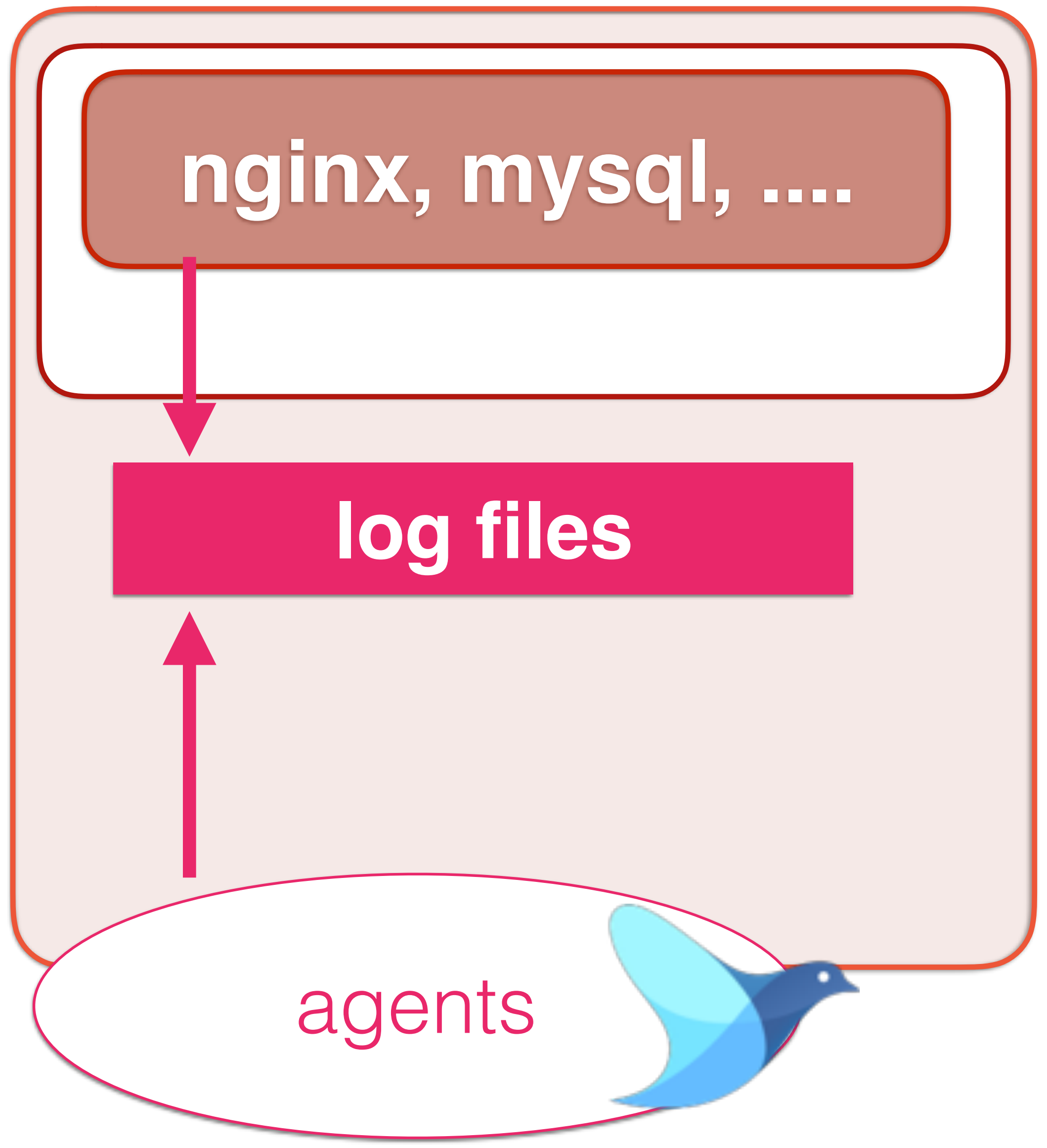
Original log:

```
the customer put an item to cart: item_id=101, items=10, client=web
```

Structured log:

```
ec_service.shopping_cart
2017-03-30 16:35:37 +0100
{
 "container_id":   "bfdd5b9....",
 "container_name": "/infallible_mayer",
 "source":         "stdout",
 "event":          "put an item",
 "item_id":        101,
 "items":          10,
 "client":         "web"
}
```

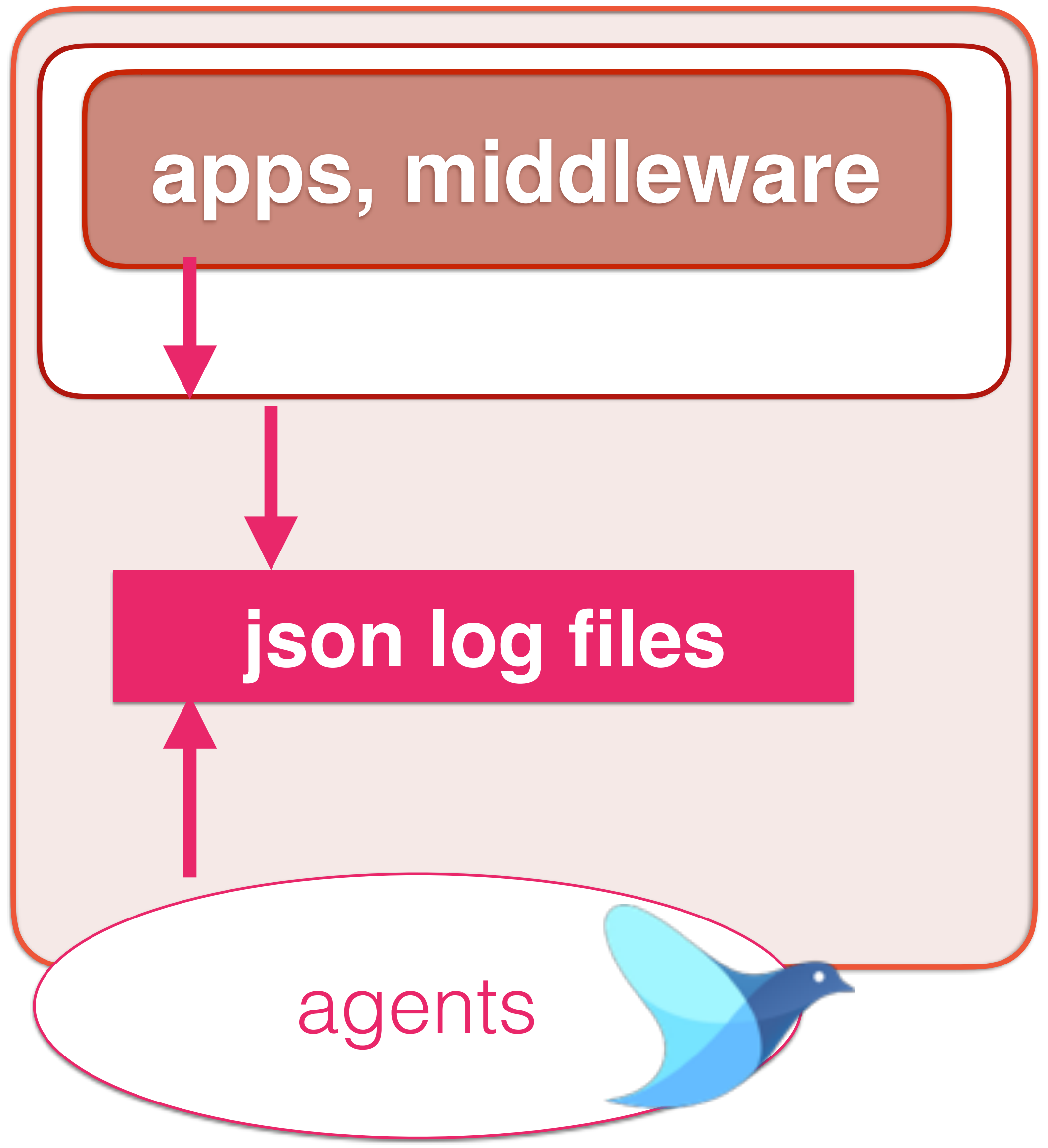
How to Ship Logs from Docker Containers



read files,
parse plain texts

Using
mounted volume

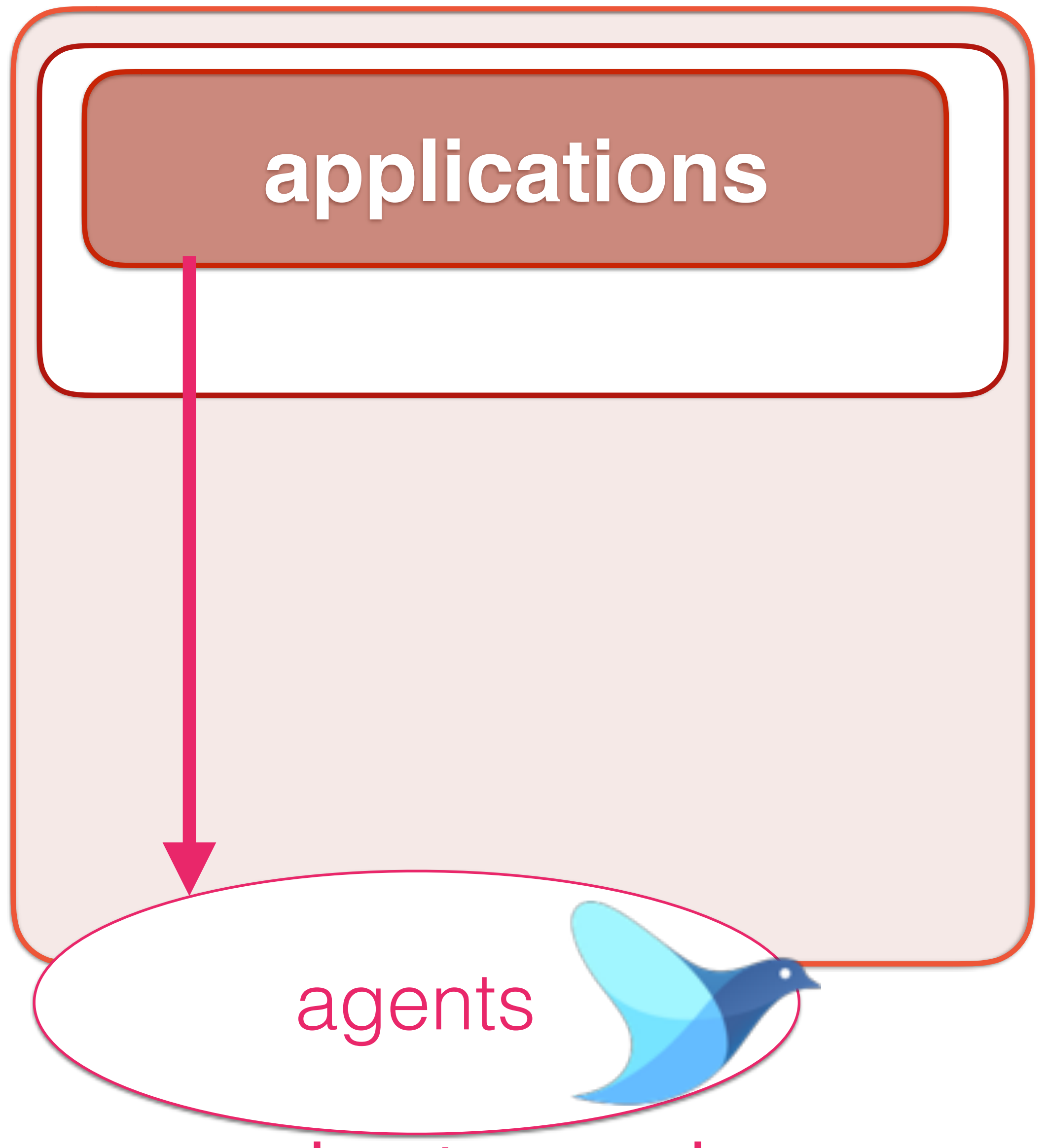
+ disk I/O penalty
+ mount points



read files,
parse json lines

Using
container json logs

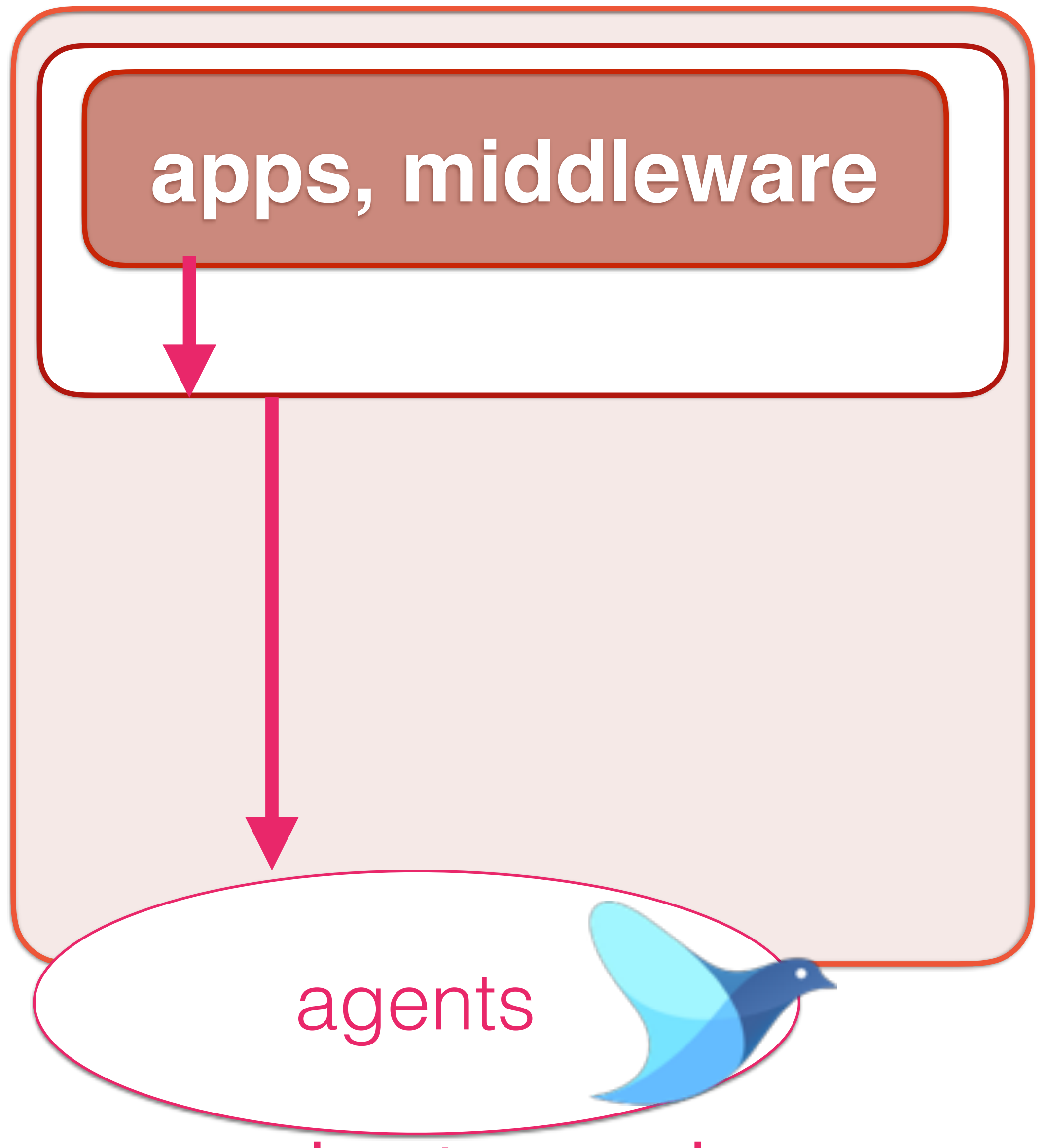
+ disk I/O penalty



just receive
transferred logs

Sending logs
to agents directly

+ logger code
+ agent config



just receive
transferred logs

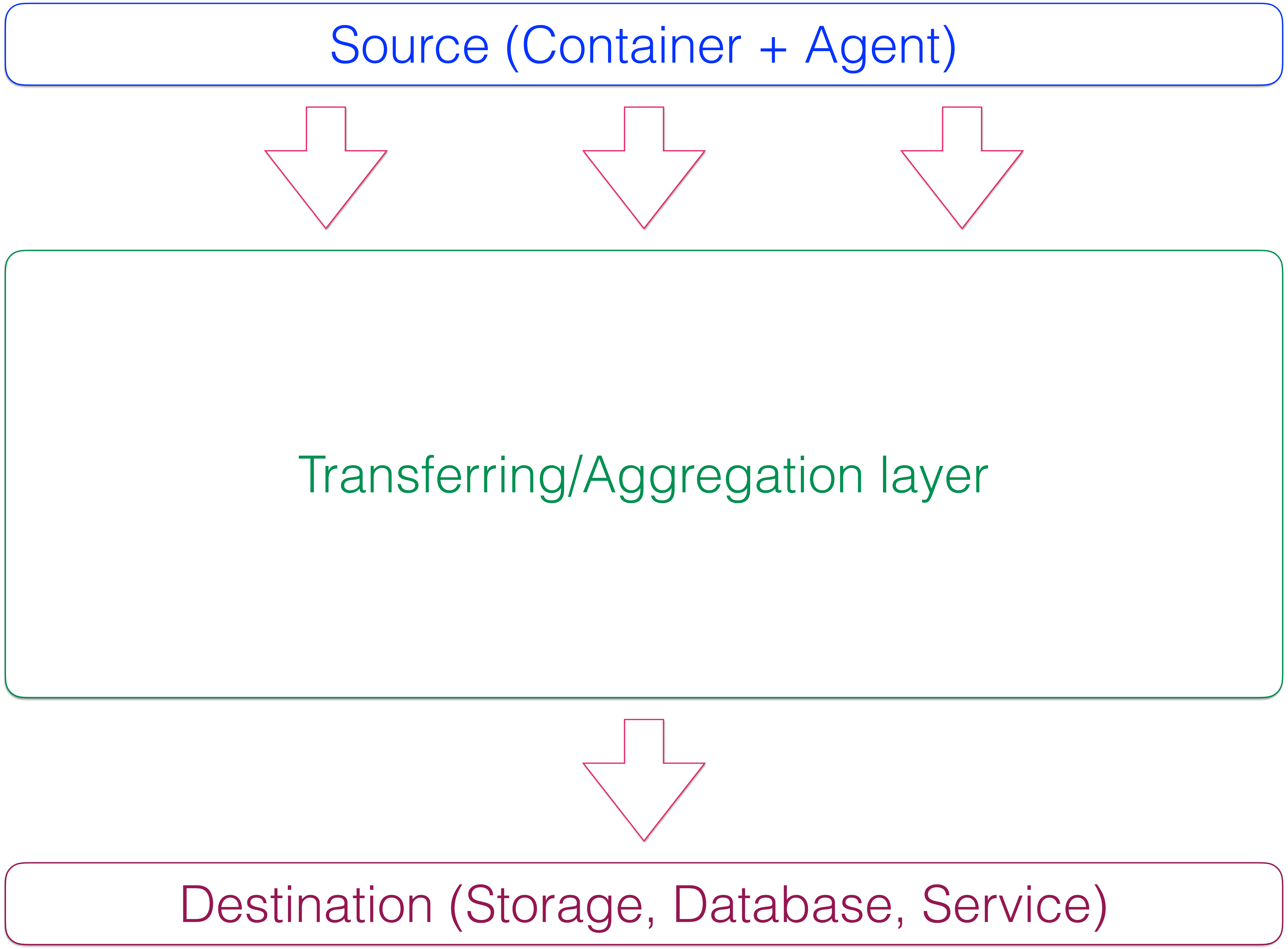
Using
logging drivers

😃

fluentd

SCALING LOGGING PLATFORM

Core Architecture: Distributed Logging



Source (Container + Agent)

Transferring/Aggregation layer

Destination (Storage, Database, Service)

Distributed Logging Workflow

Collector

- Retrieve raw logs: file system / network
- Parse log content

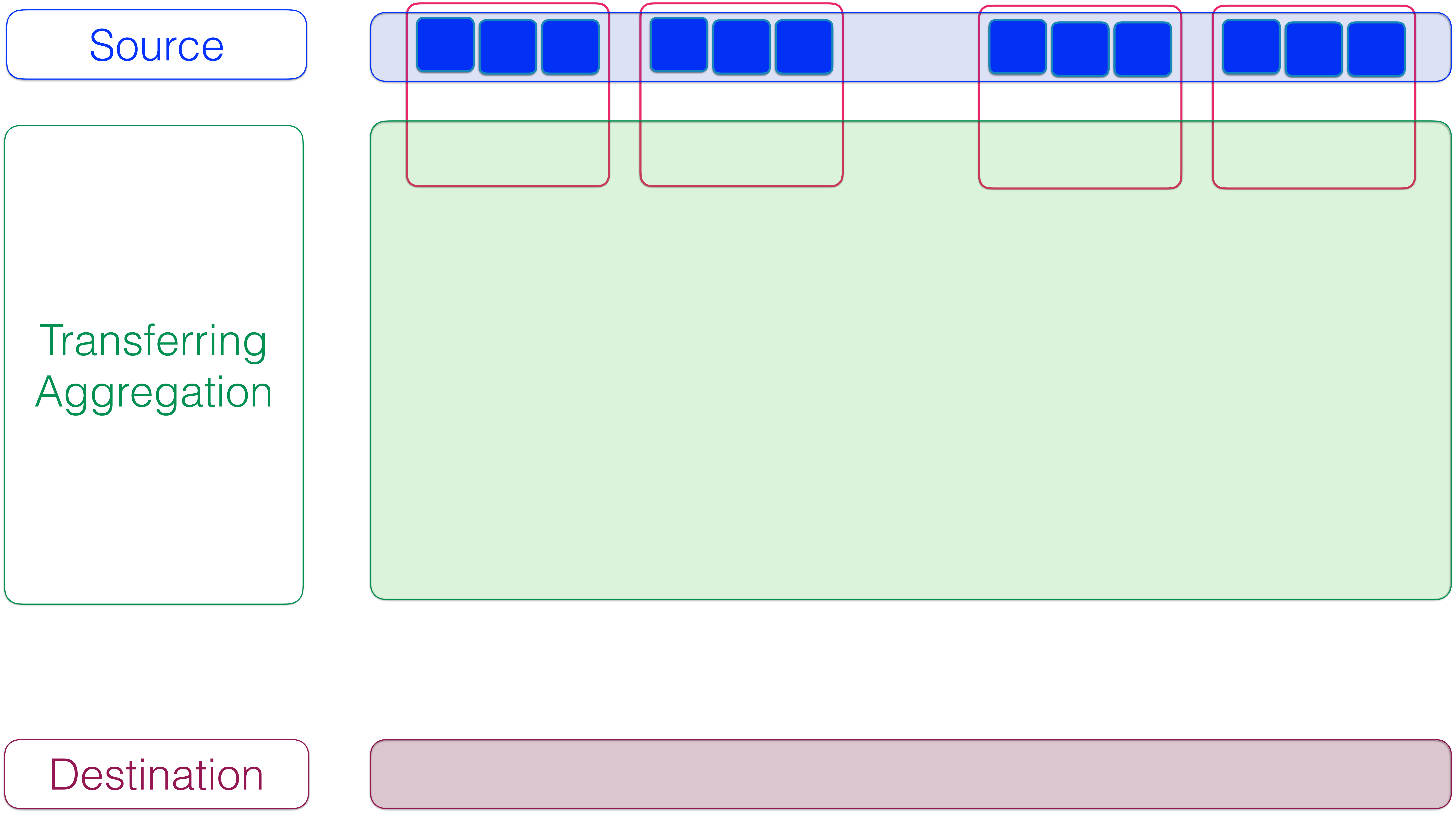
Aggregator

- Get data from multiple sources
- Split/merge incoming data into streams

Destination

- Retrieve structured logs from Aggregator
- Store formatted logs

Core Architecture: Distributed Logging



Scaling Logging

- Network Traffic
  - Split heavy log traffic into traffics to nodes
- CPU Load
  - Distribute processing to nodes about parsing/formatting logs
- High Availability
  - Switch traffic from a node to another for failures
- Agility
  - Reconfigure whole logging layer to modify destinations

PATTERNS: SOURCE/DESTINATION -SIDE AGGREGATION

Source Side Aggregation

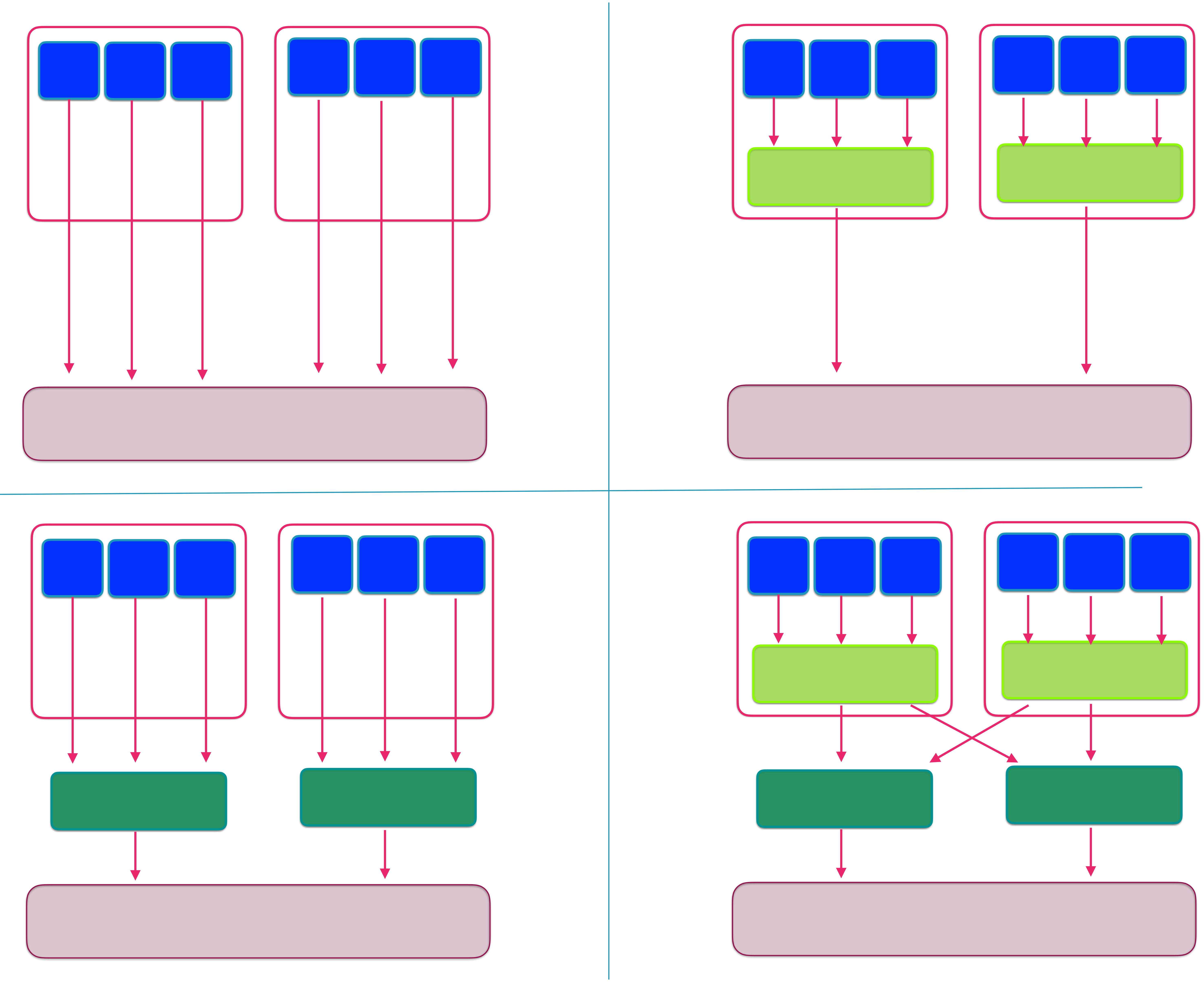
NO

YES

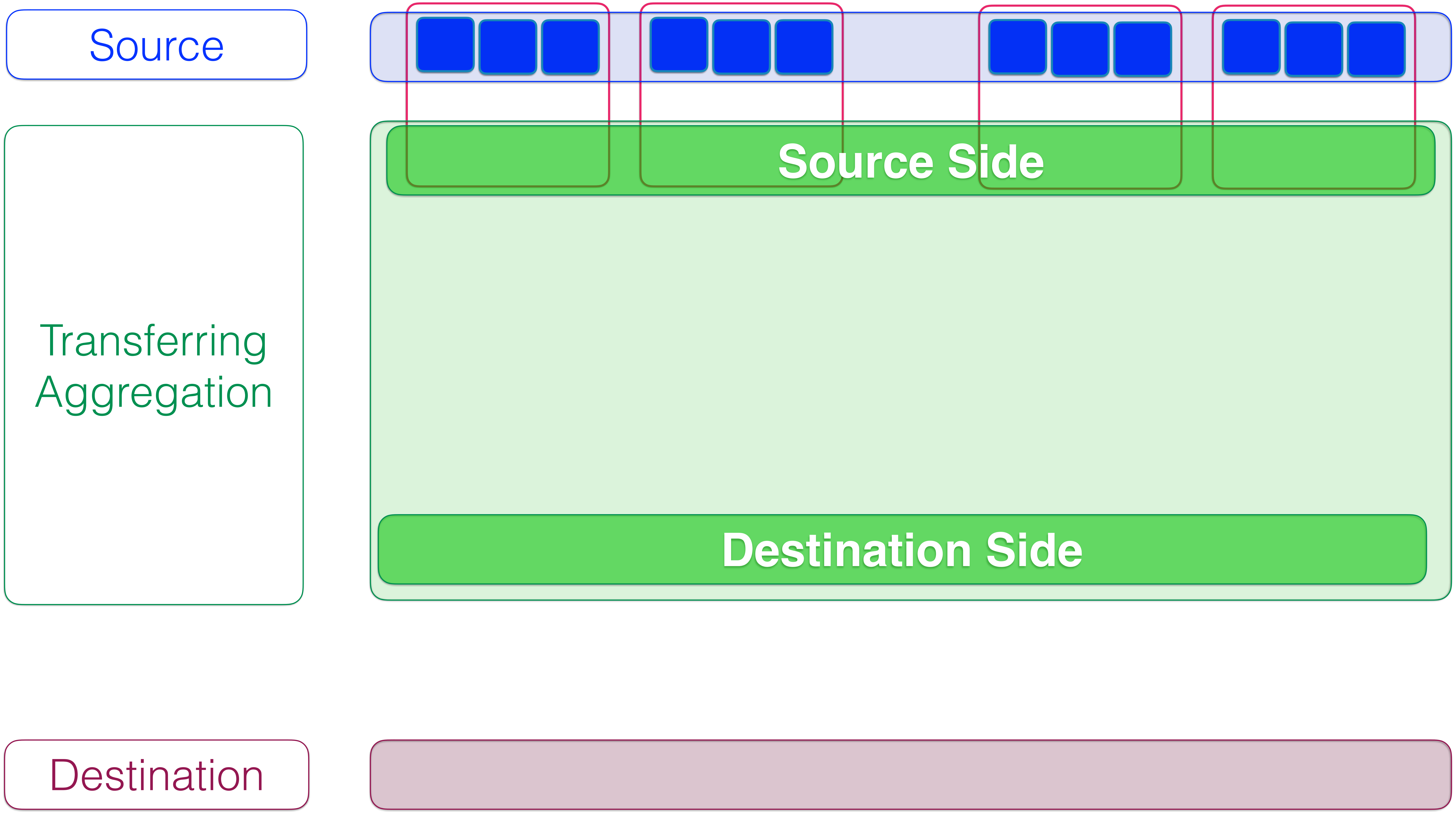
NO

Destination Side Aggregation

YES



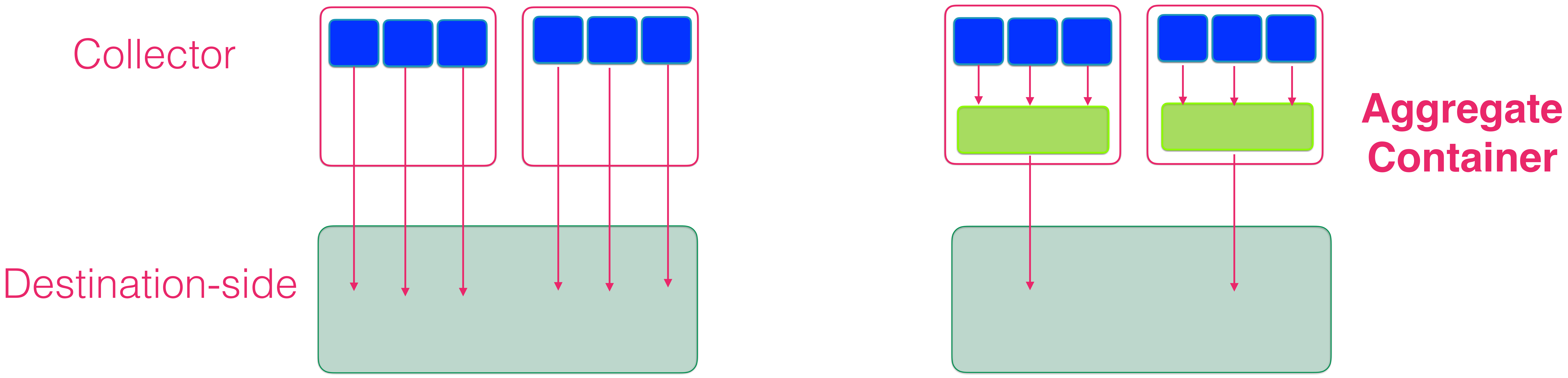
Now I'm Talking About:



Source-side Aggregation Patterns

Without Source-side Aggregation

With Source-side Aggregation



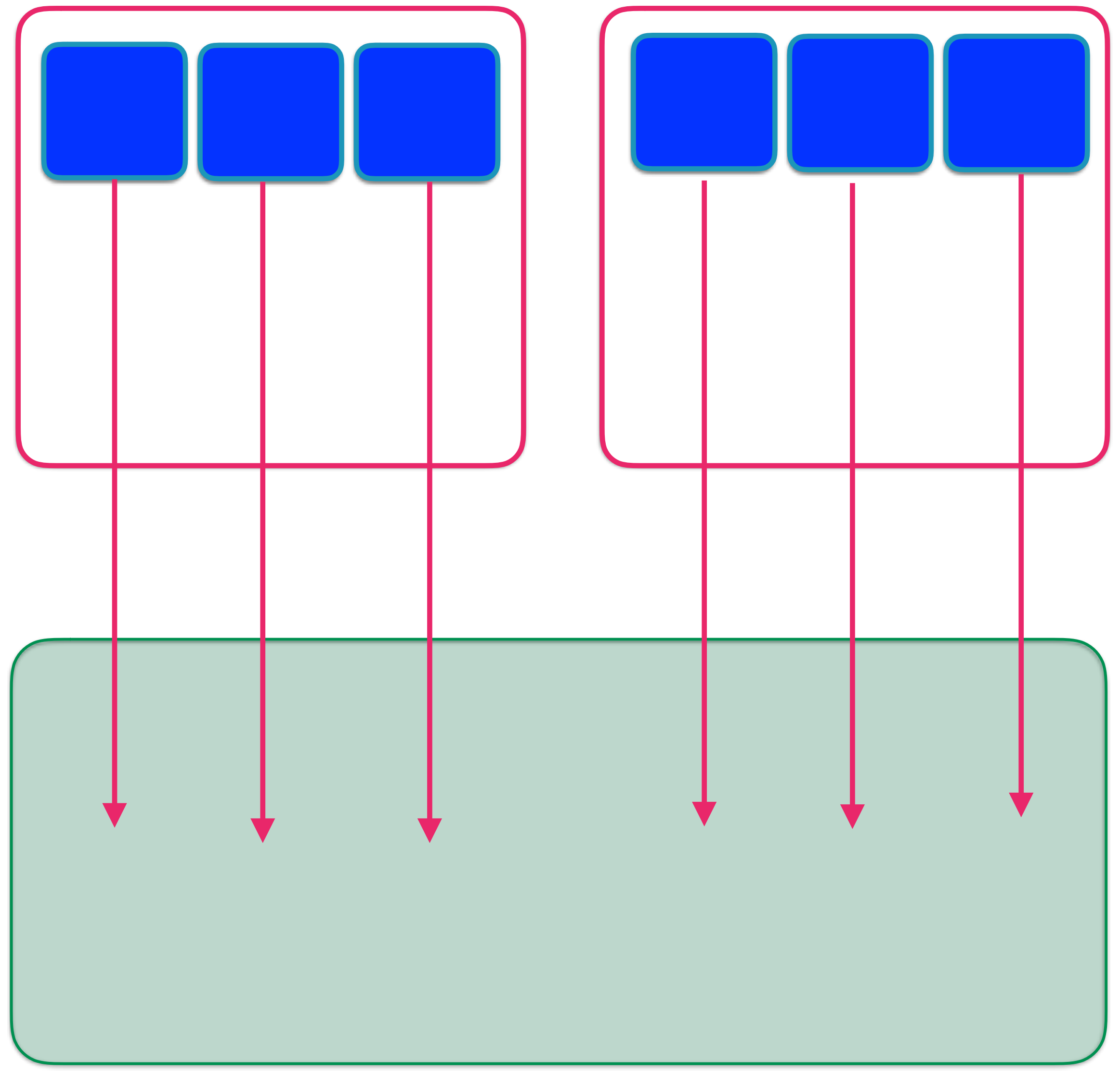
Collector

Destination-side

Aggregate Container

Aggregation Pattern without Source-side Aggregation

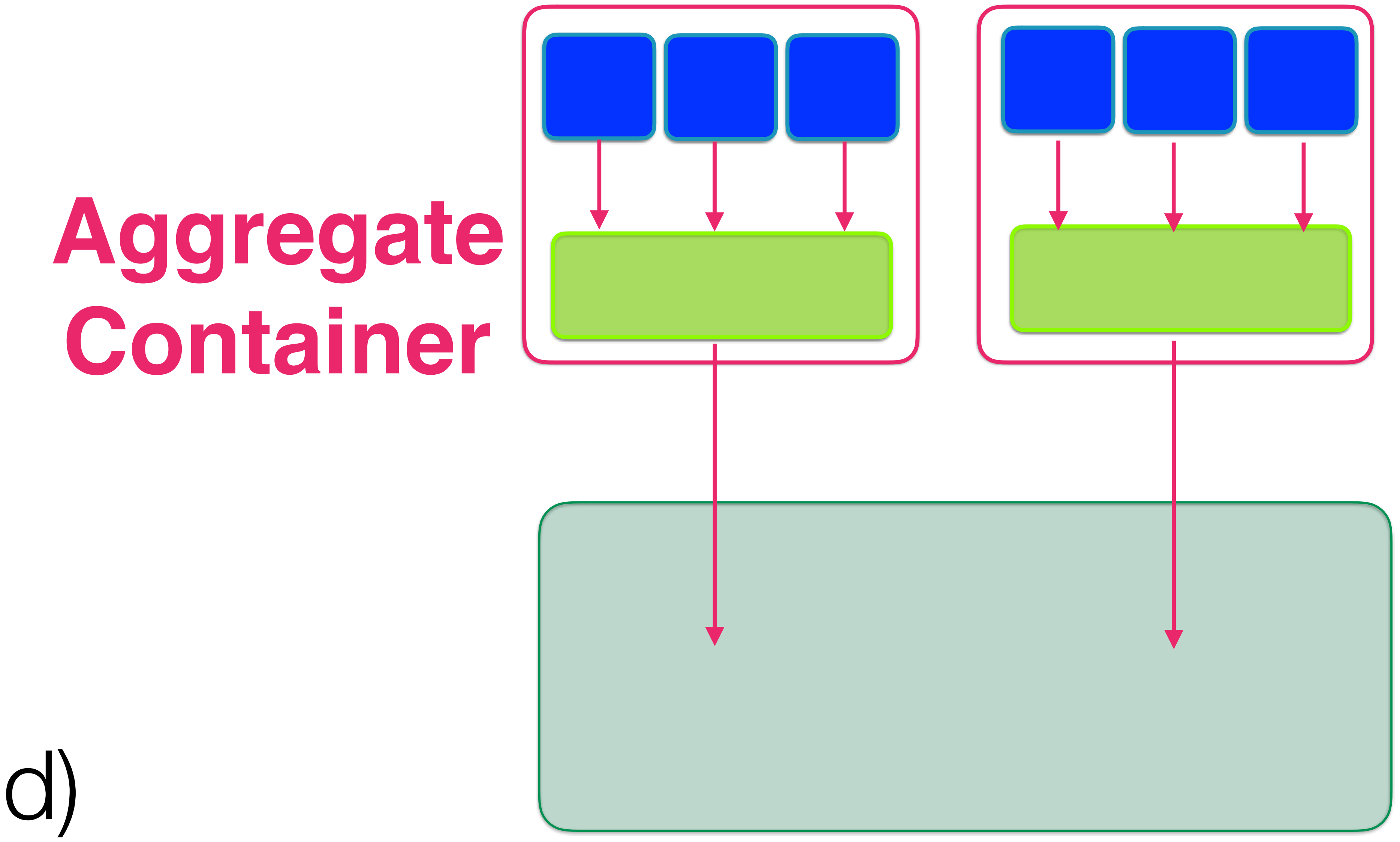
- **Pros**:
  - Simple configuration
- **Cons**:
  - Fixed aggregator (destination endpoint) address configured in containers
  - Many network connections
  - High load in aggregator / destination



Aggregation Pattern with Source-side Aggregation

- **Pros**:
  - Less connections
  - Lower load in aggregator / destination
  - Less configurations in containers
  - More agility

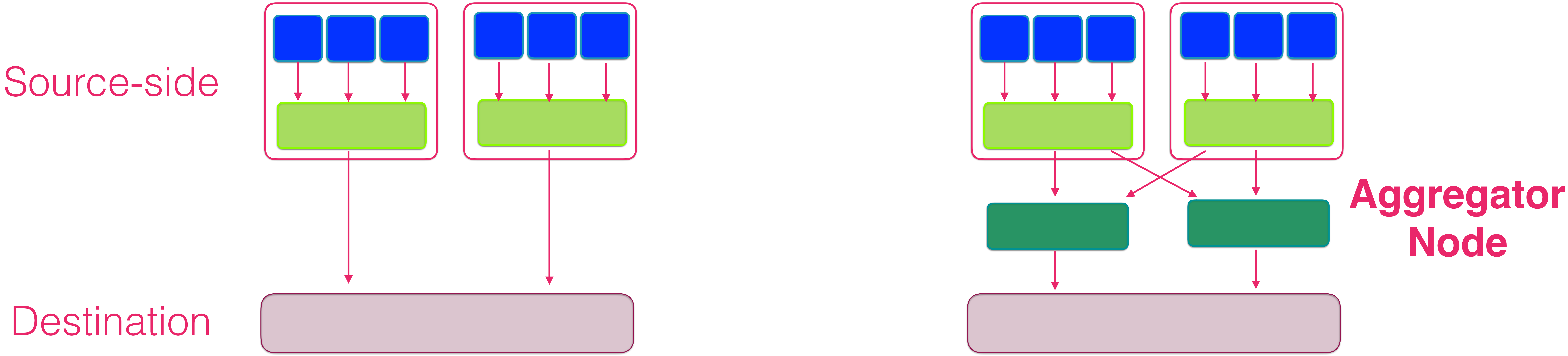
    (aggregate containers can be reconfigured)
- **Cons**:
  - Need more resources (+1 container per host)



Destination-side Aggregation Patterns

Without Destination-side Aggregation

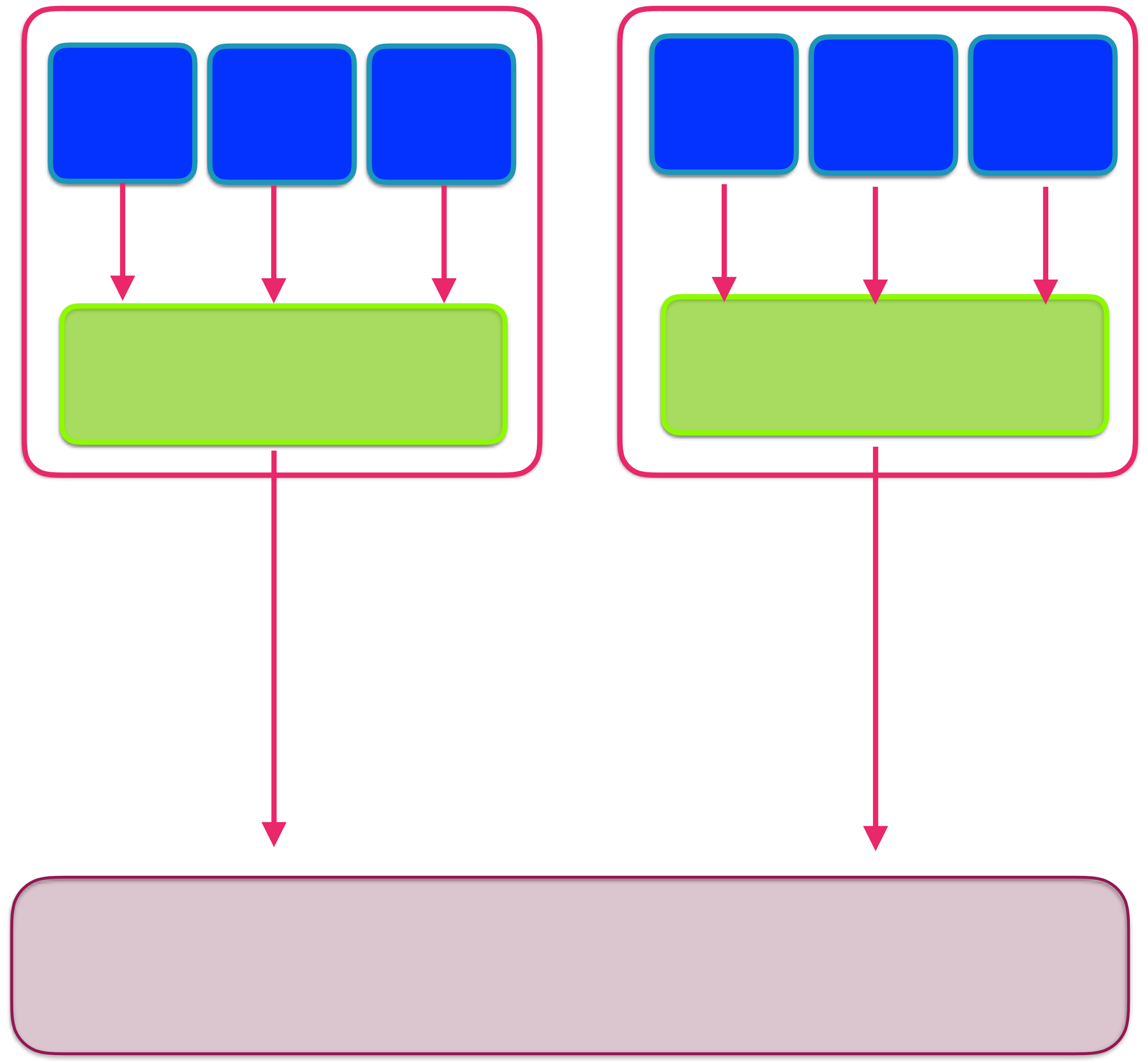
With Destination-side Aggregation



Aggregation Pattern without Destination-side Aggregation

- **Pros**:
  - Less nodes
  - Simpler configuration
- **Cons**:
  - Destination changes affects all source nodes
  - Worse performance:

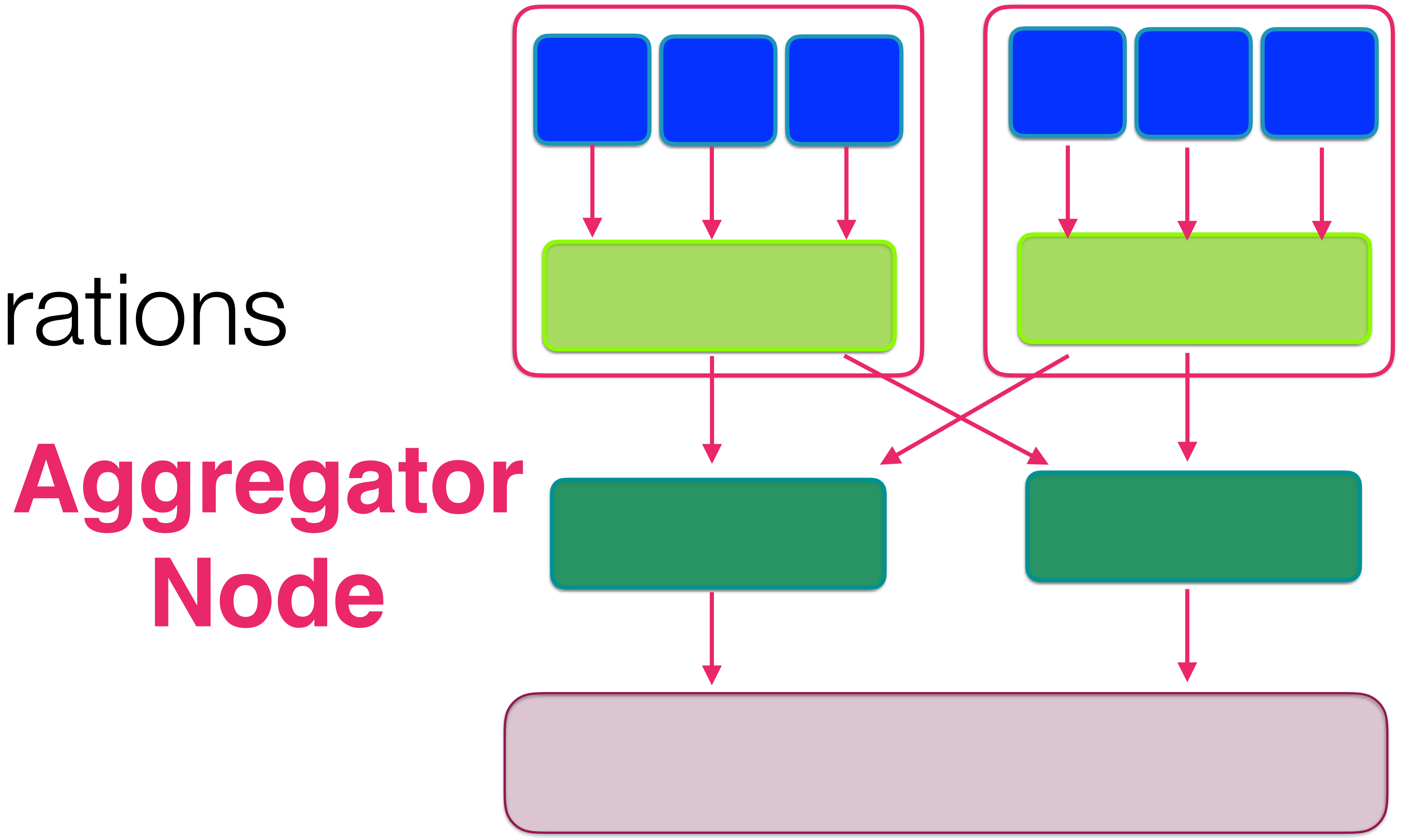
    many small write requests on destination(storage)



Aggregation Pattern with Destination-side Aggregation

- **Pros**:
  - Destination changes does NOT affect source nodes
  - Better performance:

    destination aggregator can merge write operations
- **Cons**:
  - More nodes
  - More complex configuration

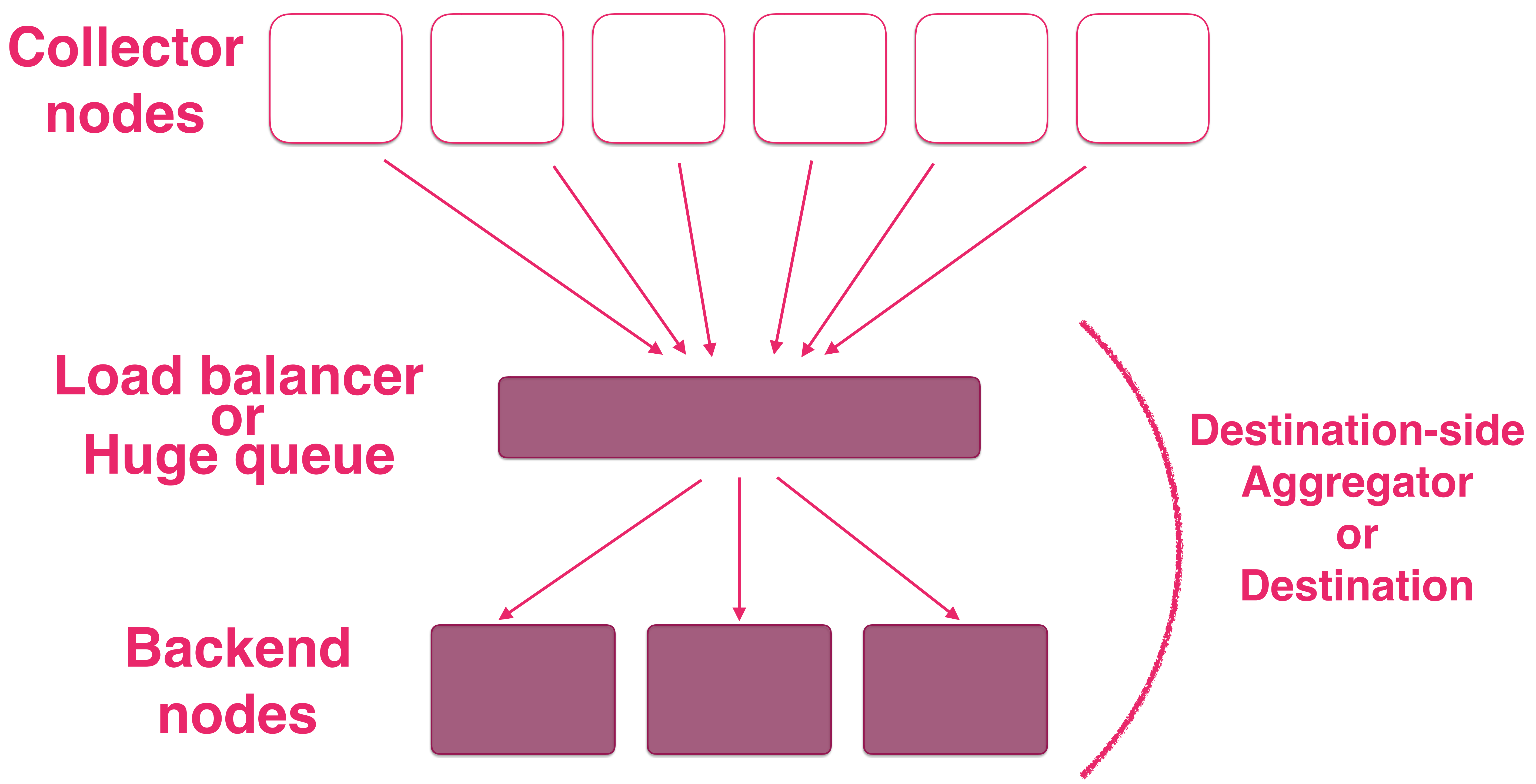


fluentd

PATTERNS:
SCALING UP/OUT DESTINATION

Scaling Destination Patterns

Scaling Up
Aggregator/Destination Endpoints



Collector nodes

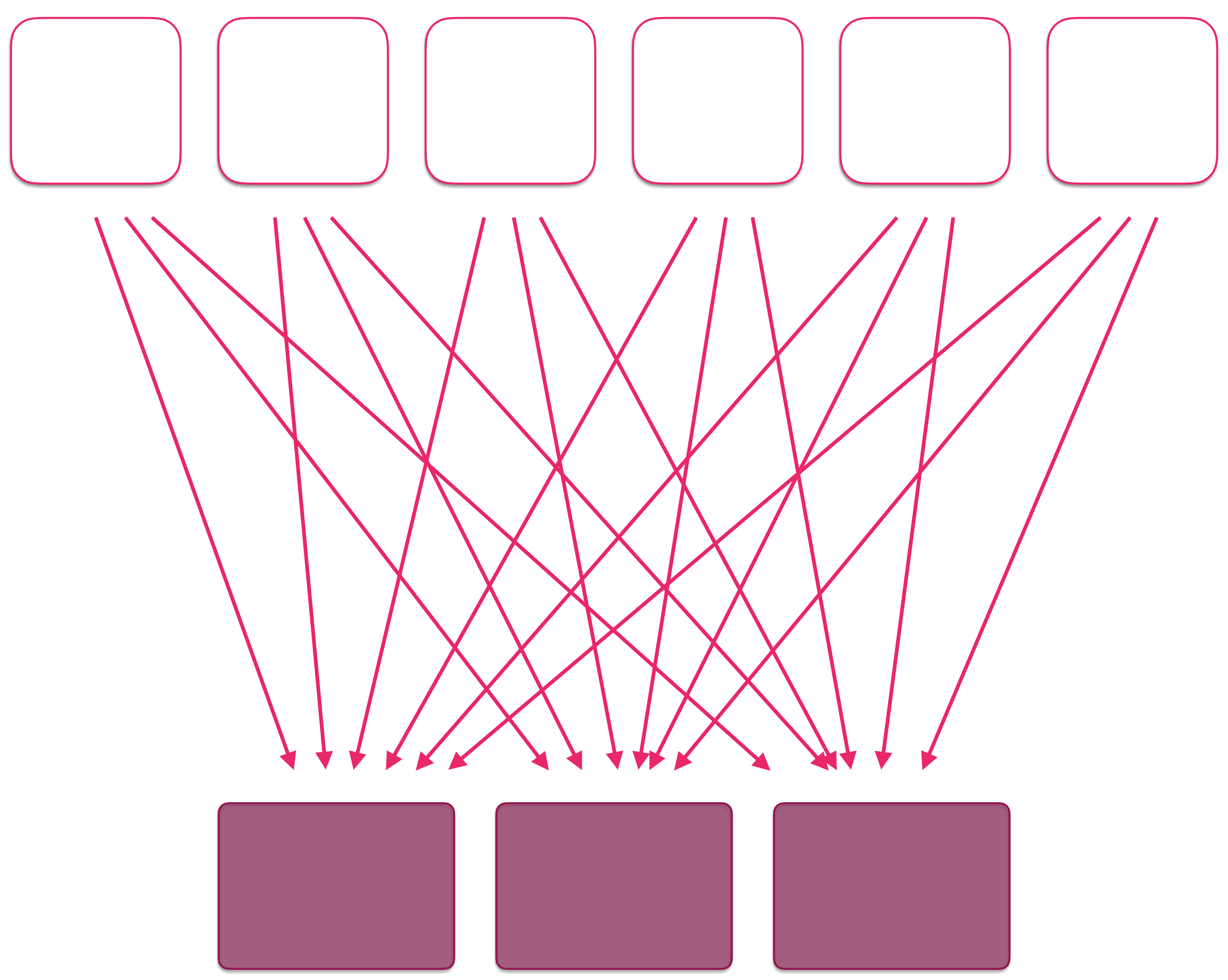
Load balancer or Huge queue

Backend nodes

Destination-side Aggregator or Destination

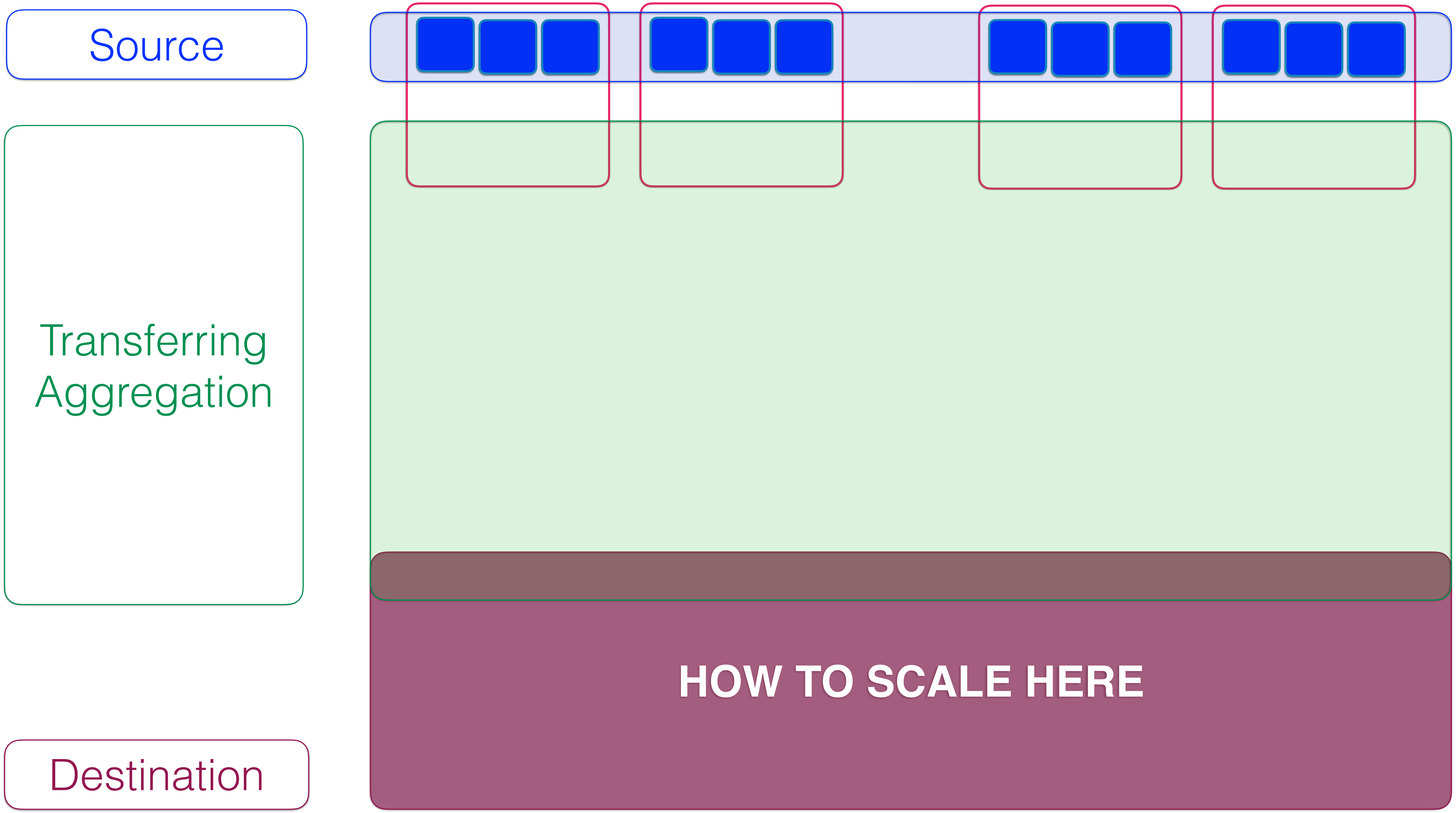
Using HTTP Load Balancer or Huge Queues

Scaling Out
Aggregator/Destination Endpoints



Using Round Robin Clients

Now I'm Talking About:



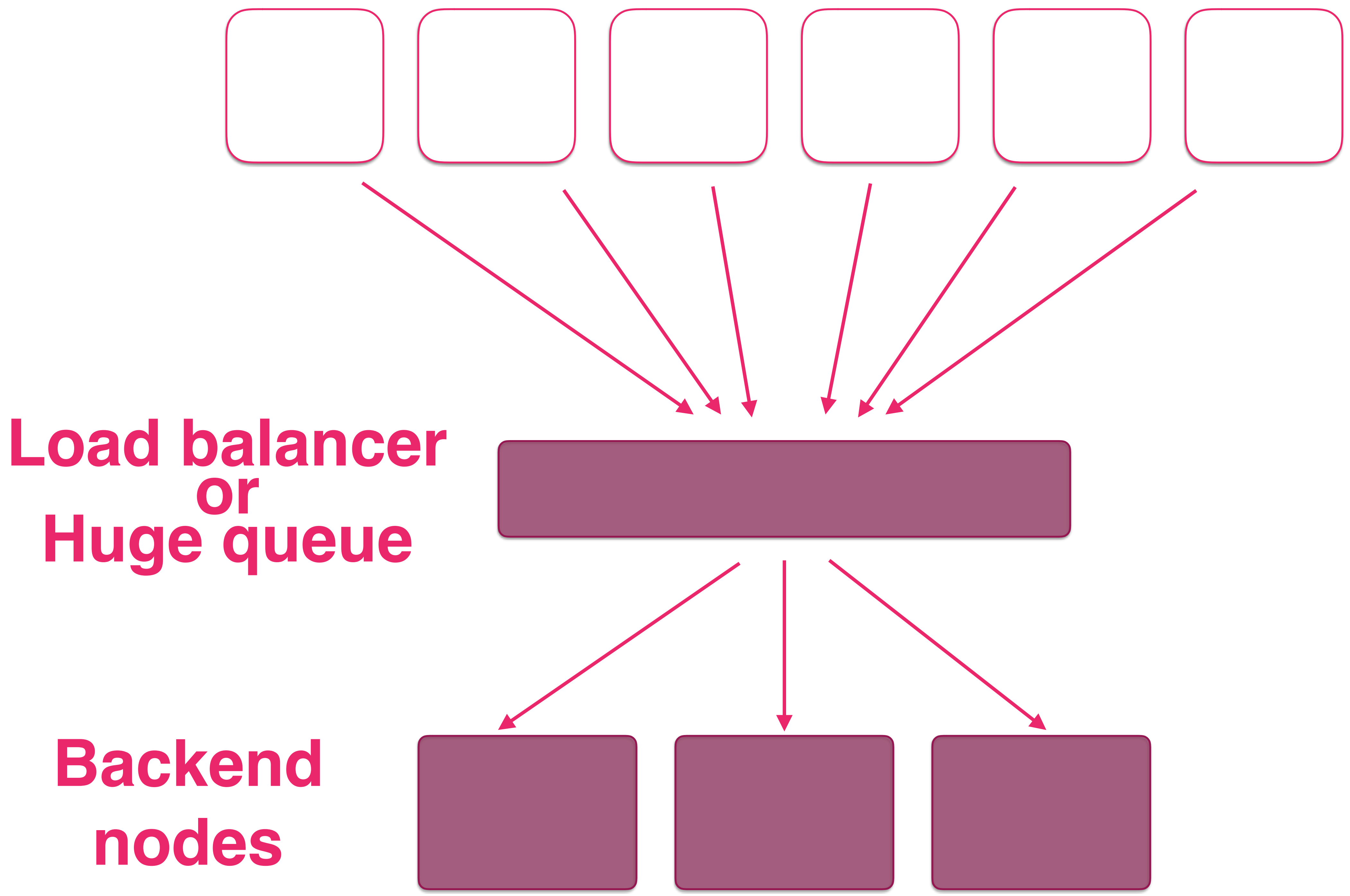
Scaling Up Destination

- **Pros**:
  - Simple configuration:

    specifying load balancer only

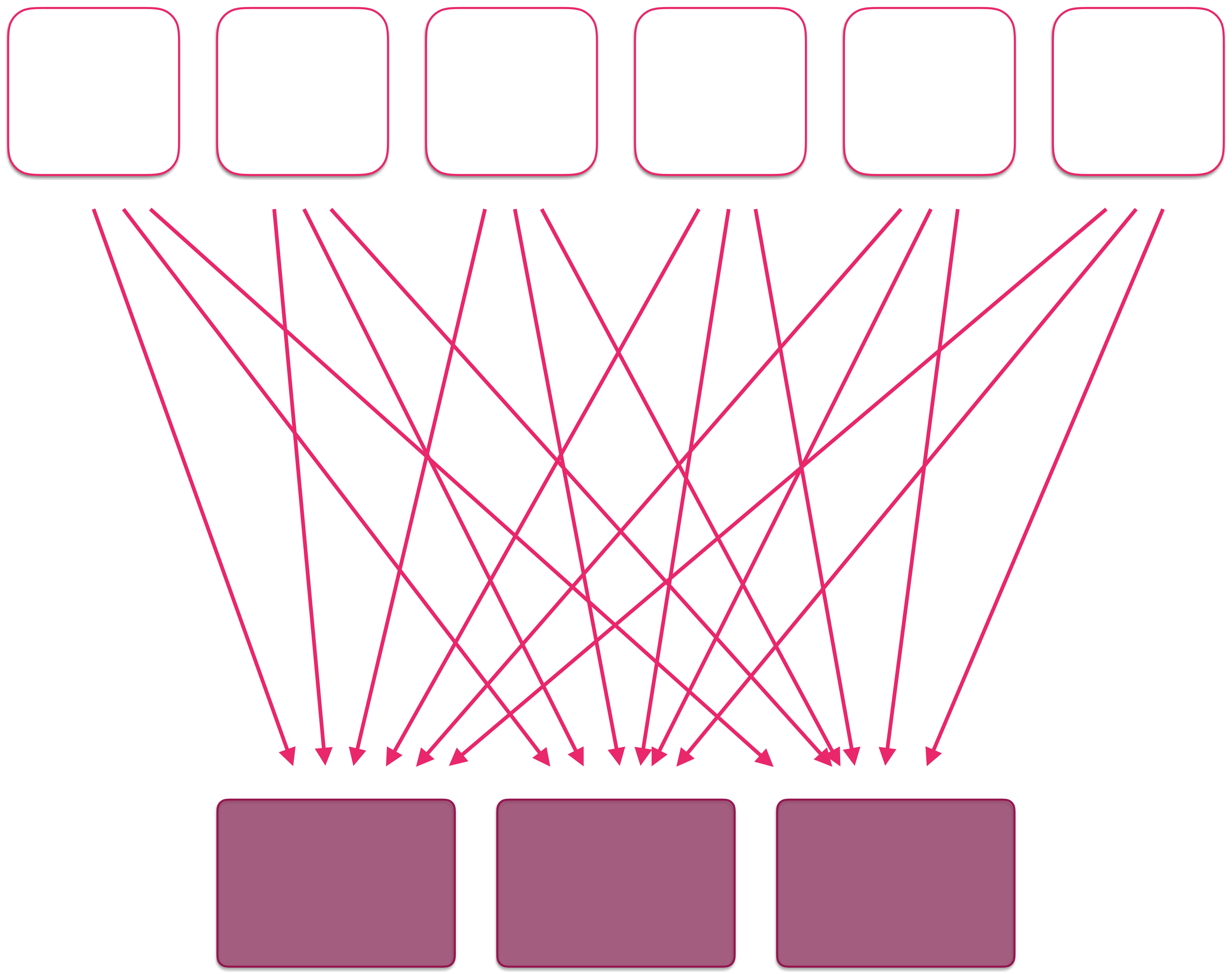
    in collector nodes
- **Cons**:
  - Upper limits about scaling up

    on Load balancer (or queue)

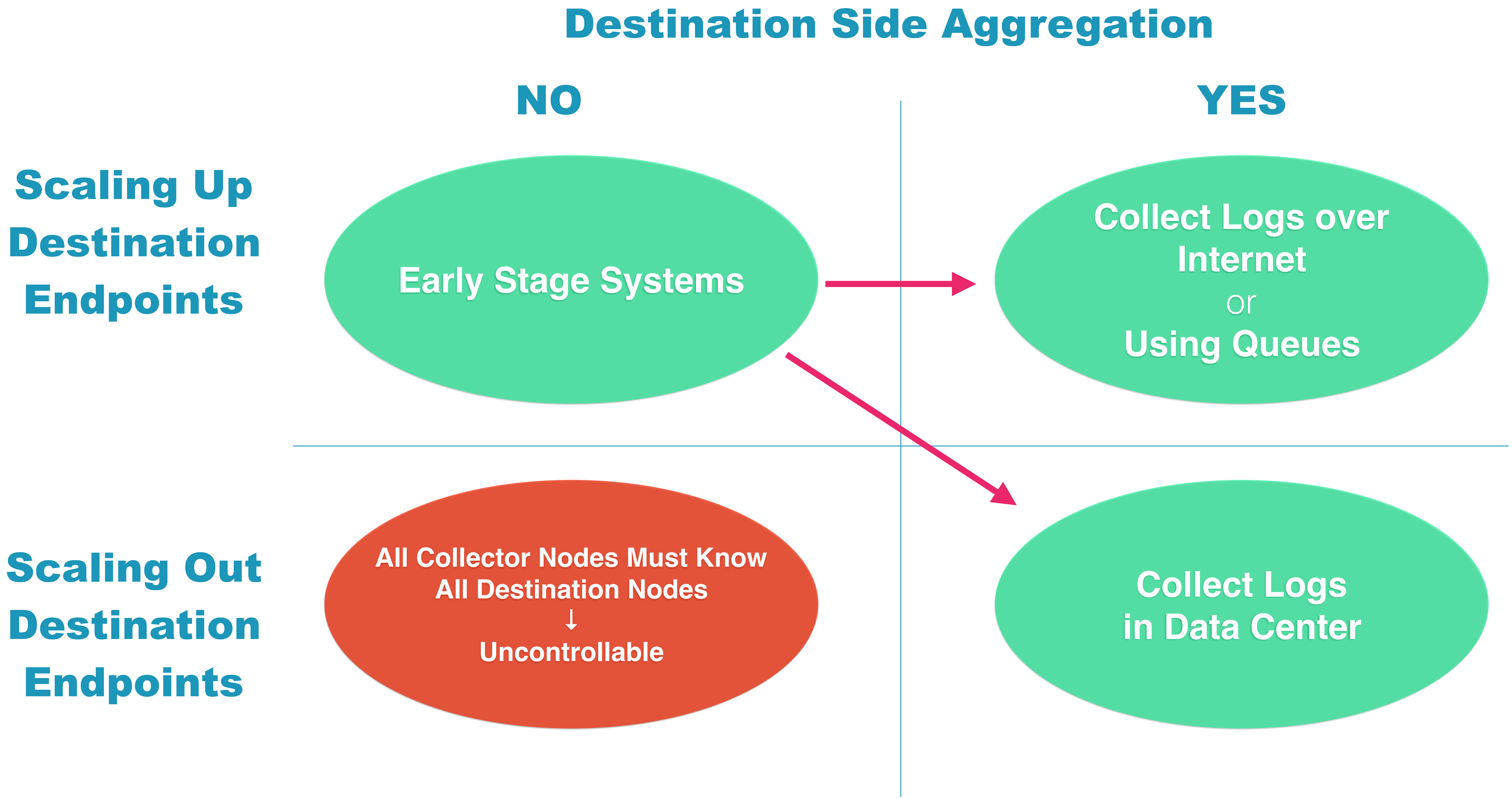


Scaling Out Destination

- **Pros**:
  - Unlimited scaling by adding nodes
- **Cons**:
  - Complex configuration in collector nodes
  - Client feature required for round-robin
  - Unavailable for traffic over Internet



Destination-side Aggregation and Destination Scaling



	Destination Side Aggregation: NO	Destination Side Aggregation: YES
Scaling Up Destination Endpoints	Early Stage Systems	Collect Logs over Internet or Using Queues
Scaling Out Destination Endpoints	All Collector Nodes Must Know All Destination Nodes ↓ Uncontrollable	Collect Logs in Data Center

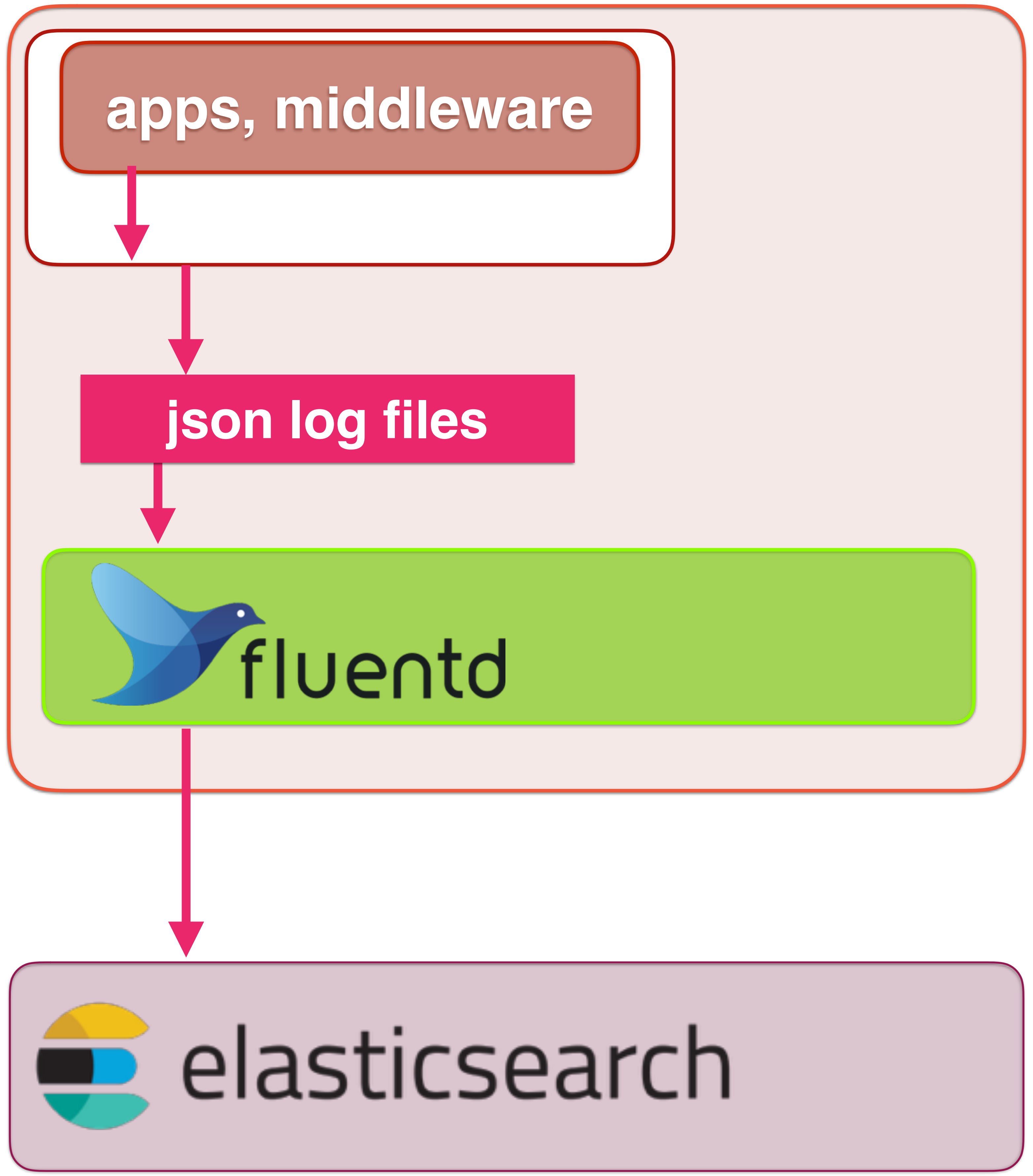
PRACTICES

Practices: Docker + Fluentd

- **Docker Fluentd Logging Driver**
  - Docker containers can send these logs to Fluentd directly, with less overhead
- Fluentd's **Pluggable Architecture**
  - Various destination systems (storage/database/service) are available by changing configuration
- **Small Memory Footprint**
  - Source aggregation requires +1 container per hosts: less additional resource usage is fine!

Practice 1: Source-side Aggregation + Scaling Up

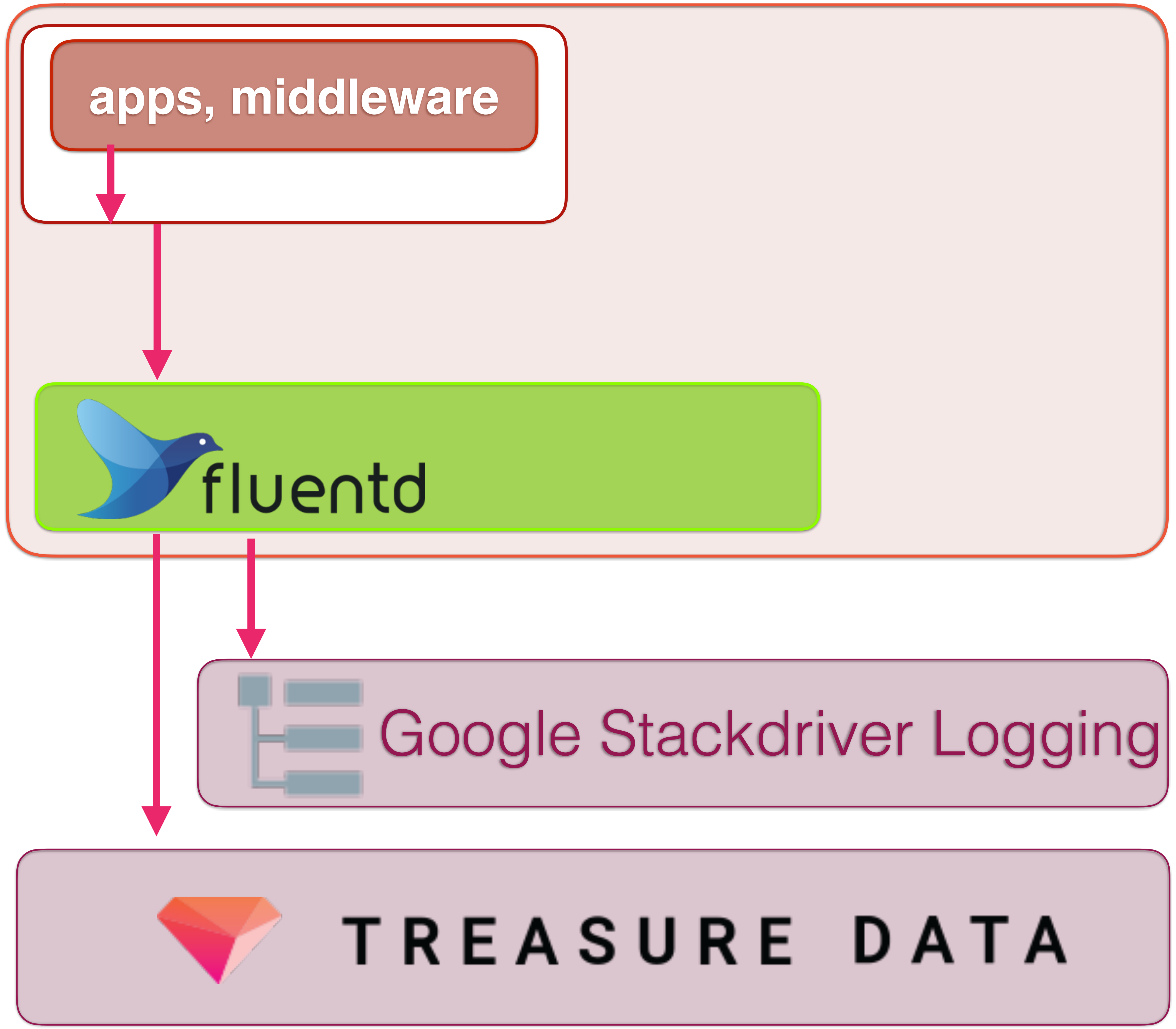
- Kubernetes: Fluentd + Elasticsearch
  - a.k.a EFK stack (inspired by ELK stack)
    - Elasticsearch - Fluentd - Kibana



https://kubernetes.io/docs/tasks/debug-application-cluster/logging-elasticsearch-kibana/

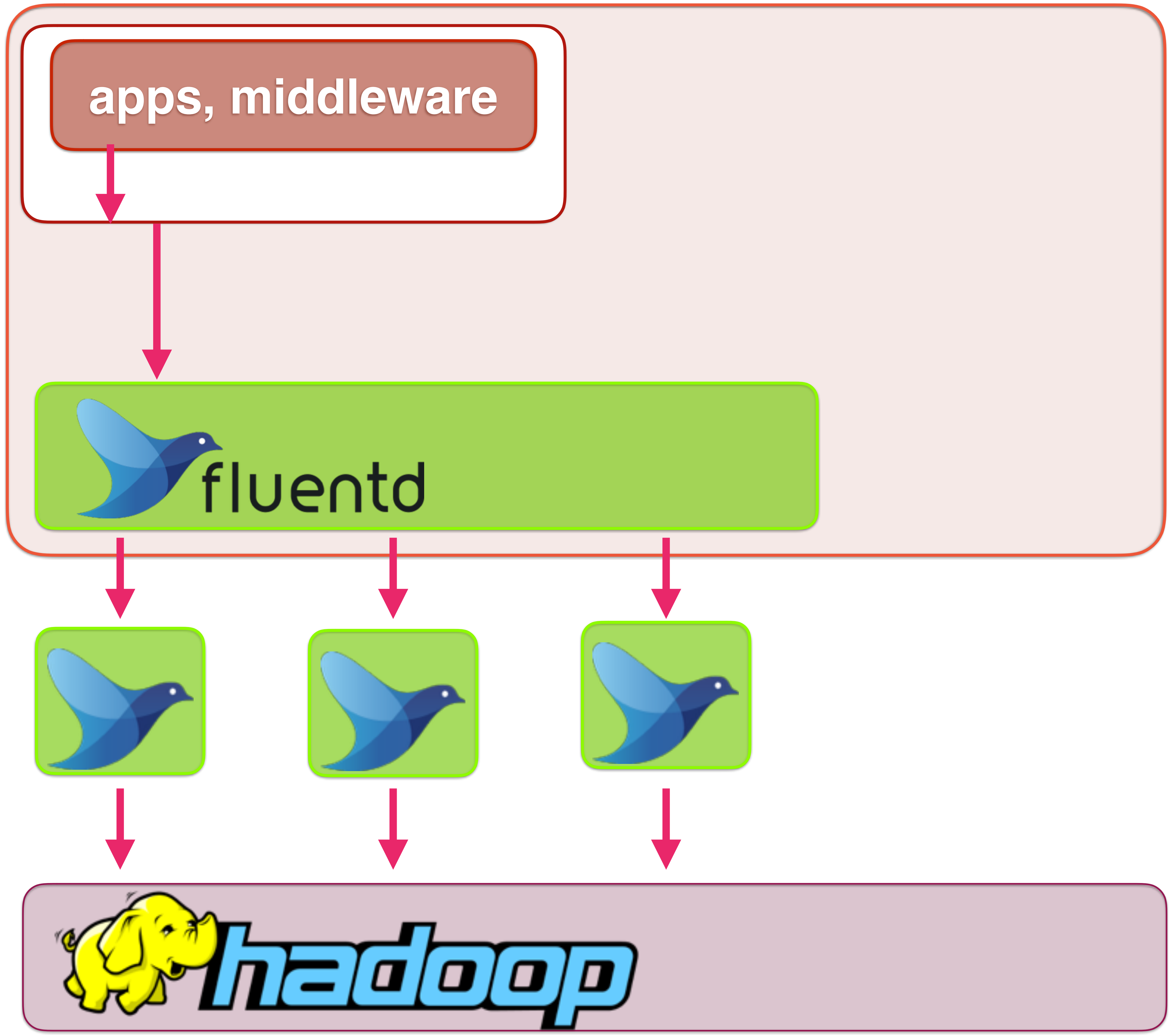
Practice 2: Source-side Aggregation + Scaling Up

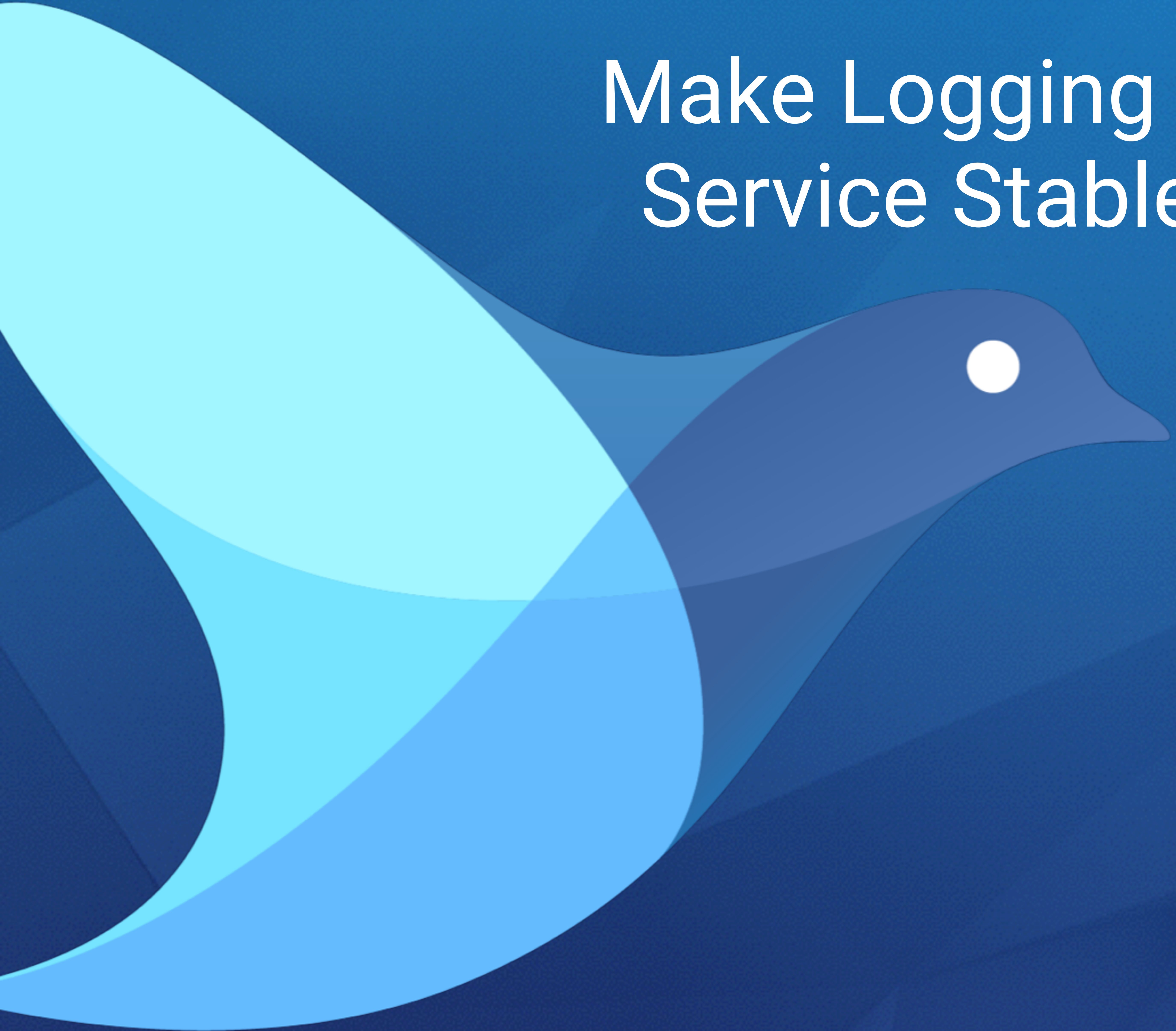
- Containerized Applications
  - w/ Google Stackdriver for Monitoring
  - w/ Treasure Data for Analytics



Practice 3: Source/Destination-side Aggregation + Scaling Out

- Containerized Application
  - w/ Log processing on Hadoop
  - writing files on HDFS via WebHDFS
- Hadoop HDFS prefers large files on HDFS:
  - Destination-side aggregation works well





Make Logging Scalable,
Service Stable & Business Growing.

Happy Logging!
@tagomoris