SEAT GEEK

How SeatGeek Successfully Handle High Demand Ticket On-Sales

Vitor Pellegrino
Director of Engineering

Anderson Parra
Senior Software Engineer

Anderson Parra

Senior Software Engineer

@anderparra

/anderparra

Vitor Pellegrino

Director of Engineering

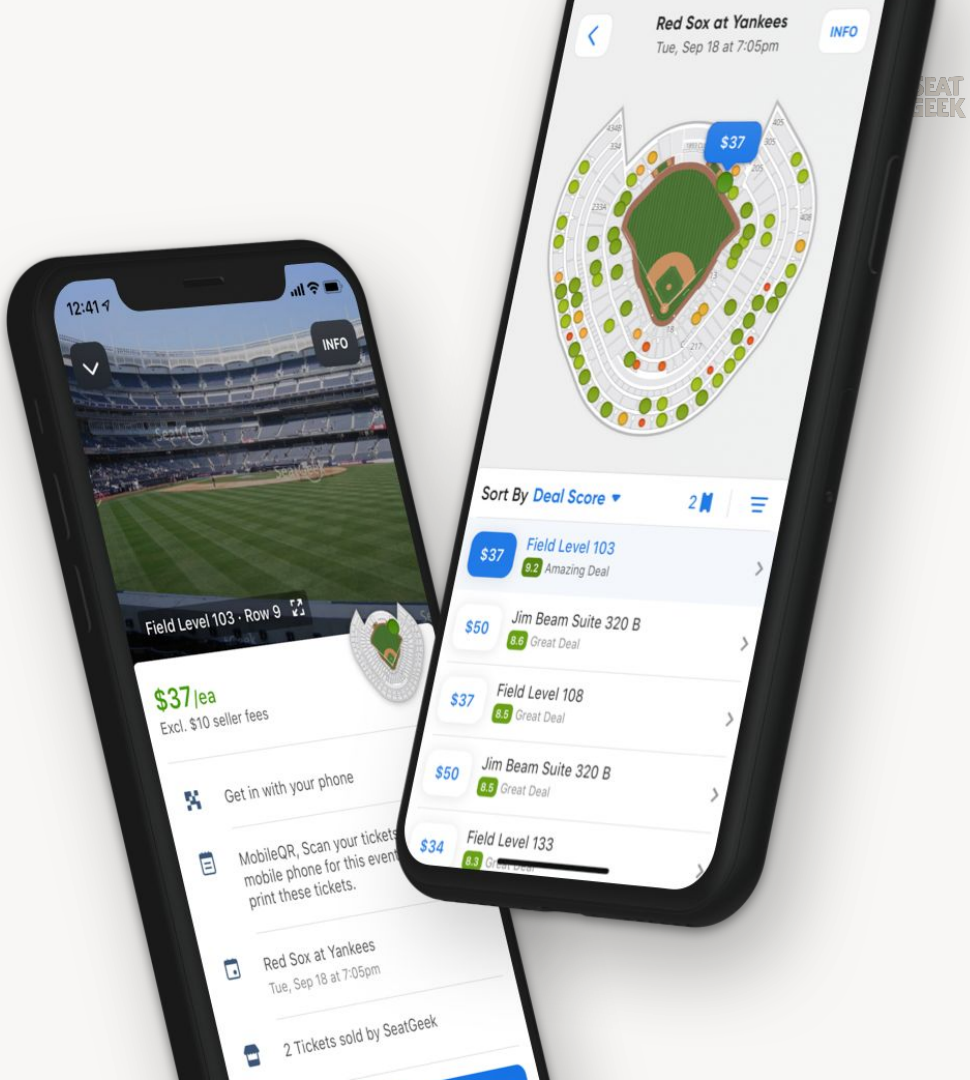
@pellegrino

/vitorpellegrino

What's SeatGeek?

With our combination of technological prowess, user-first attitude and dashing good looks, we, **SeatGeek**, are simplifying and modernizing the ticketing industry.

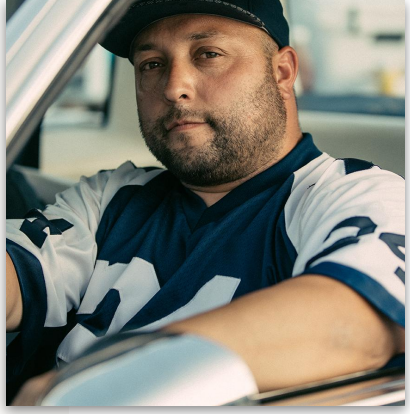
By simultaneously catering to both the consumer and enterprise markets, we're powering a new, open entertainment industry where fans have effortless access to experiences, and teams, venues and shows have seamless access to their audiences. We think it's time that everyone can expect more from ticketing.

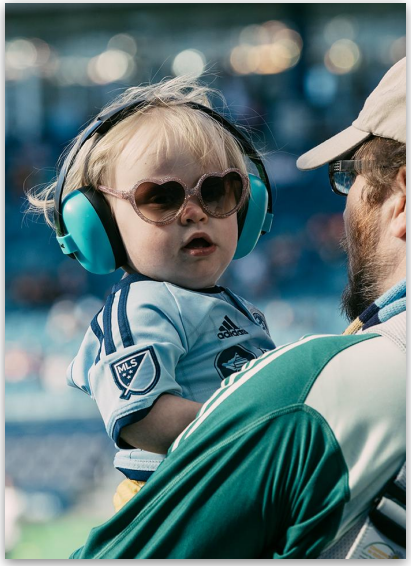


SeatGeek

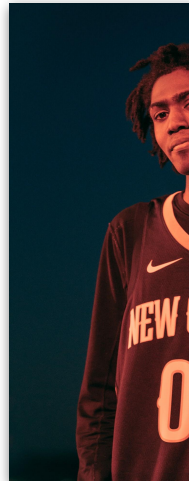
Vision Statement

Become the **largest and most loved** ticket marketplace by offering consumers a meaningfully **better ticketing experience**.



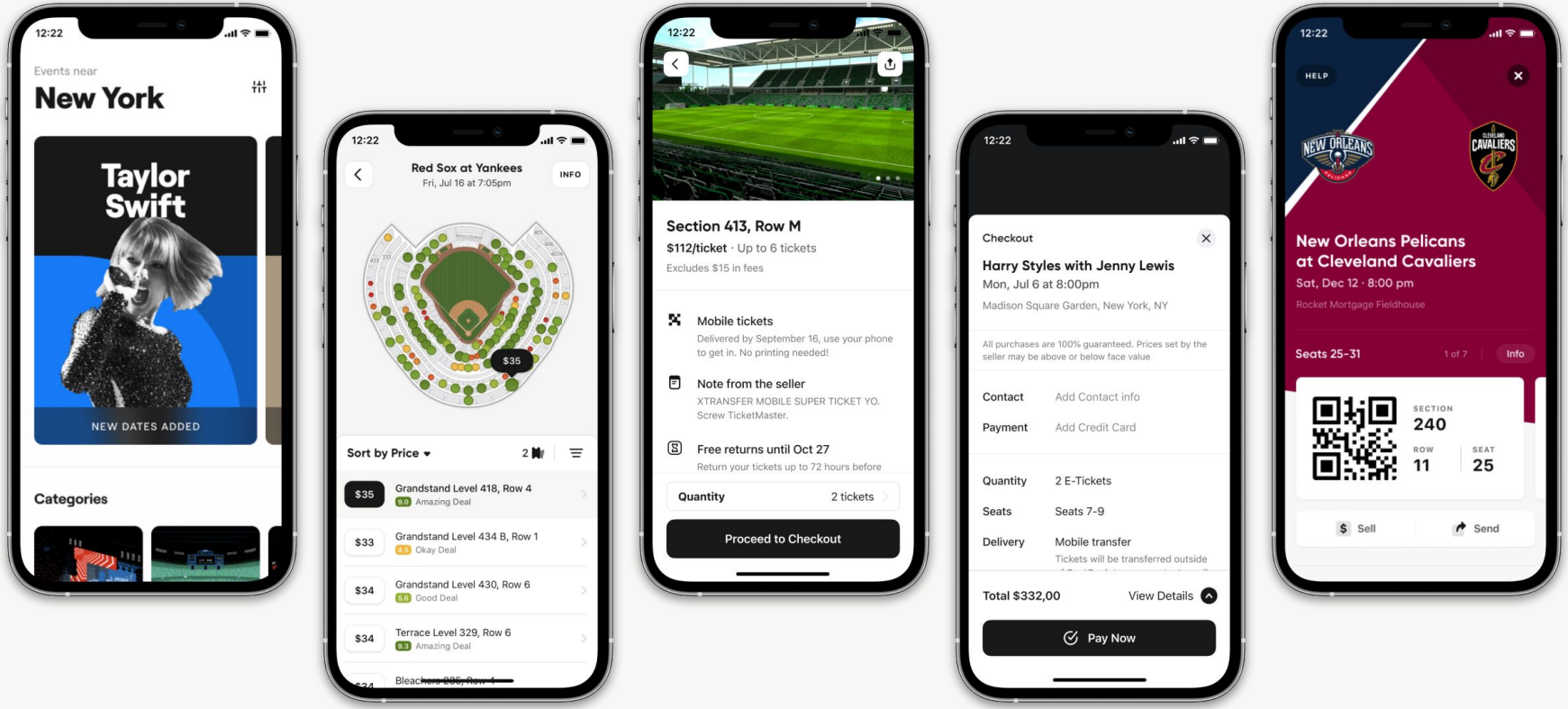




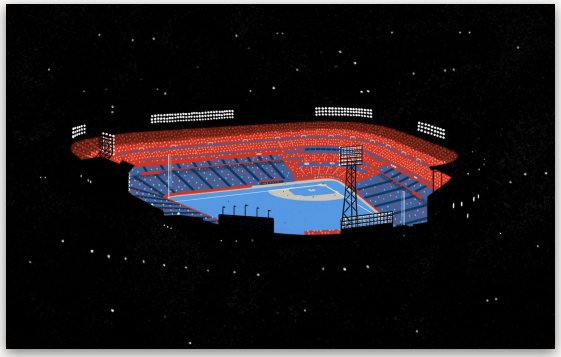

SEAT
GEEK

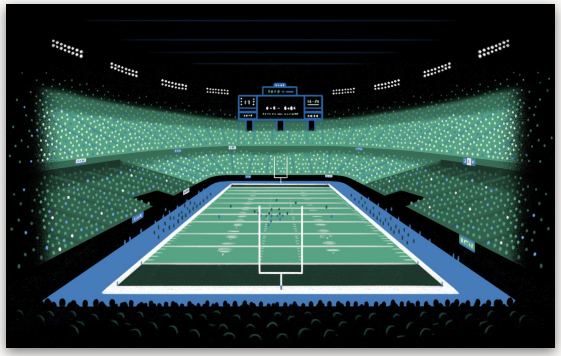
The Ticketing Problem

Who is the customer anyway?

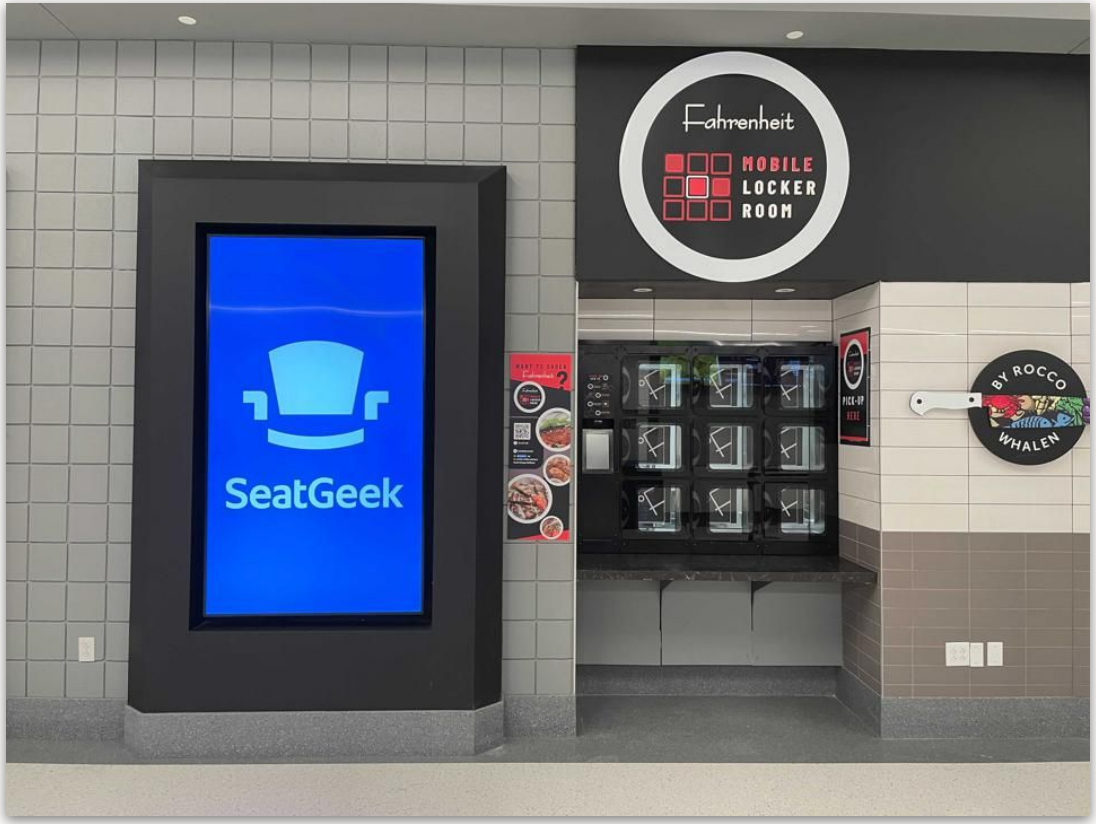


Who is our customer anyway?



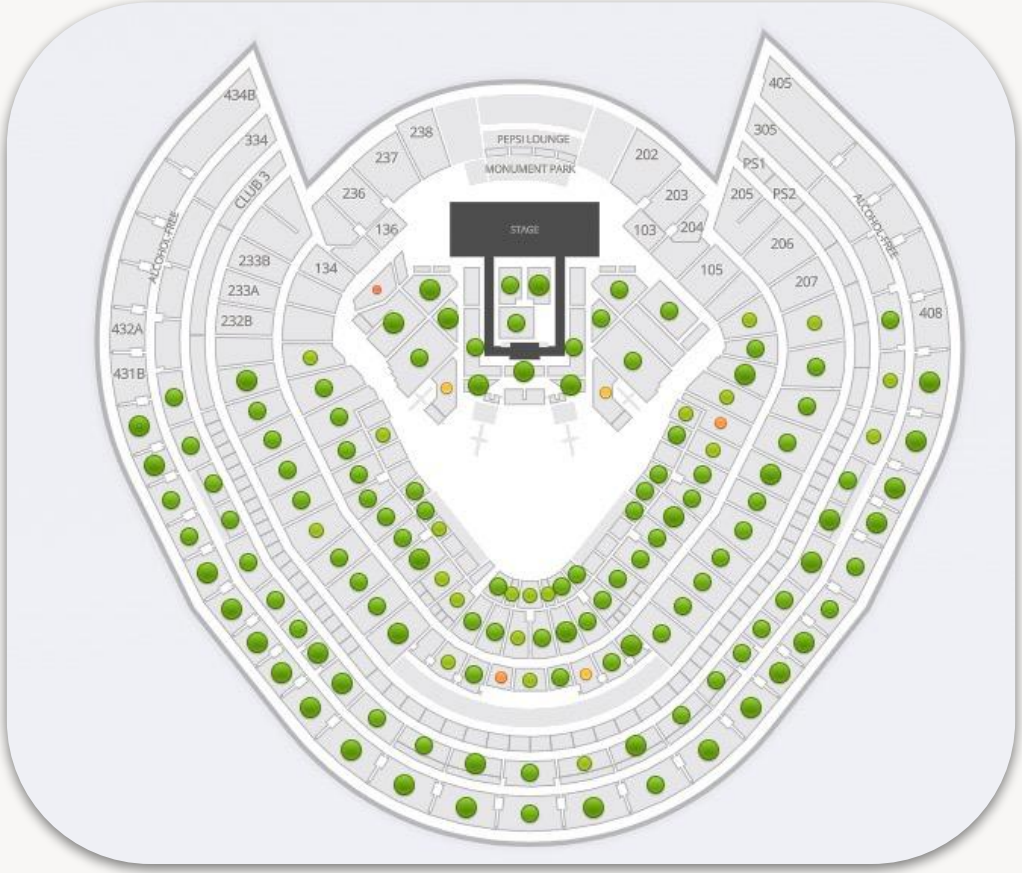


Different interfaces



https://www.forbes.com/sites/shlomosprung/2021/04/21/cleveland-cavaliers-seatgeek-bring-mobile-contactless-food-ordering-to-rocket-mortgage-fieldhouse

Ticketing Problem



Normal operations

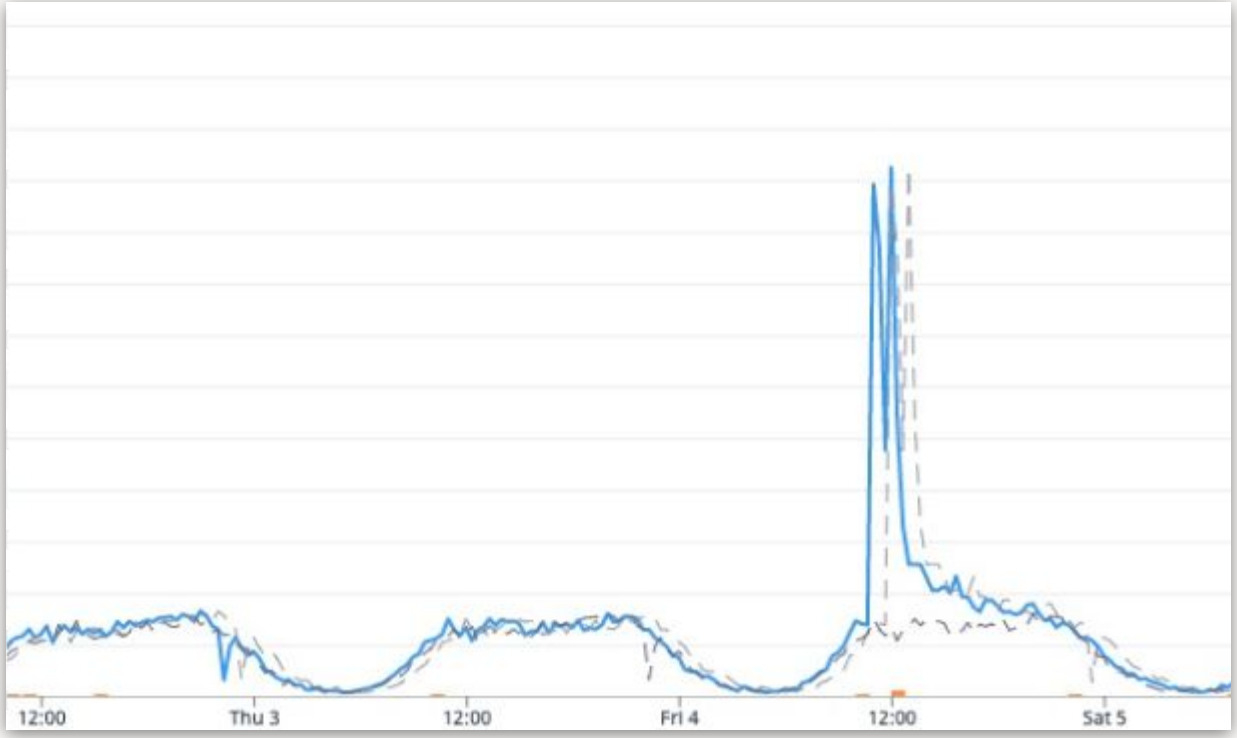
SEAT GEEK

Normal operations

vs

On-sales

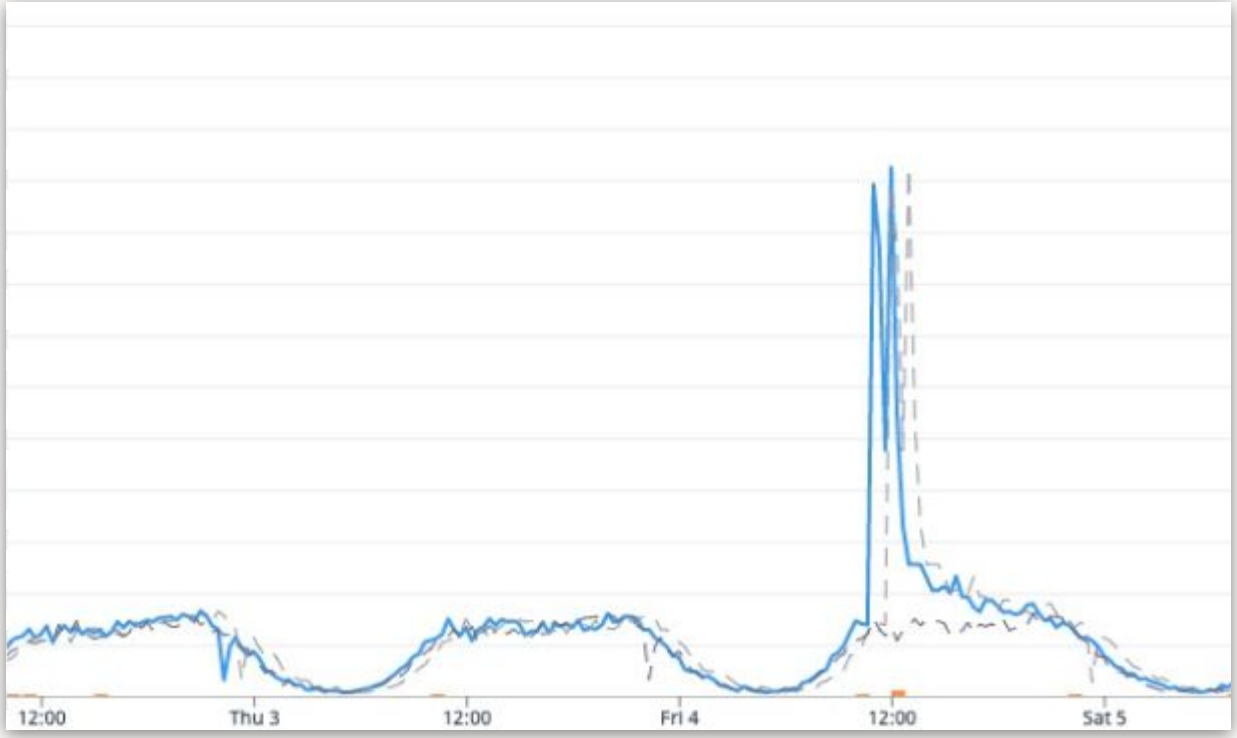
Normal operations vs on-sales



Normal operations vs on-sales

Autoscale is

not enough



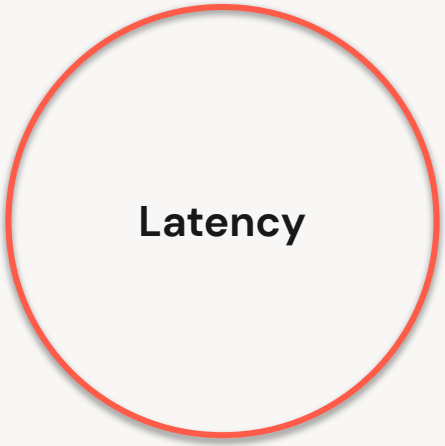
Normal operations vs on-sales

SEAT GEEK

Tradeoffs
might **change**

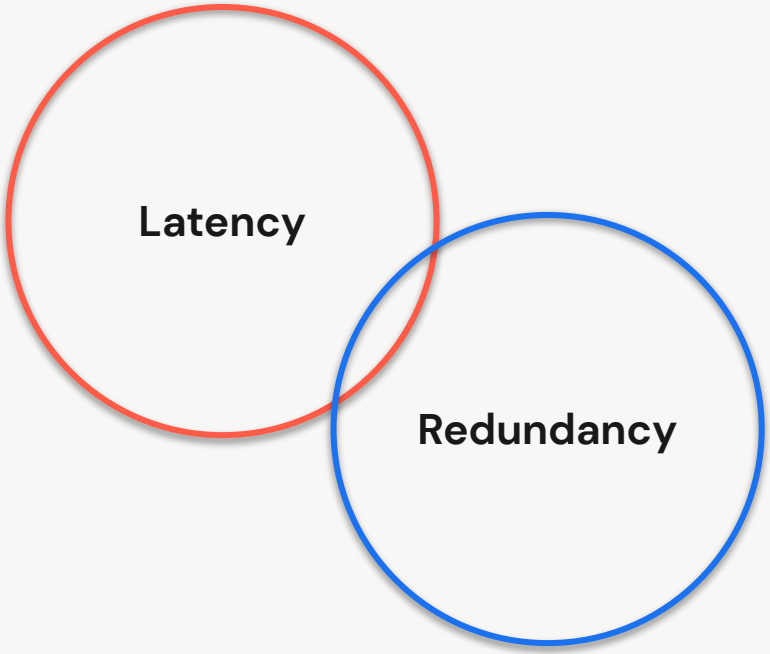
Normal operations vs on-sales

Tradeoffs
might **change**



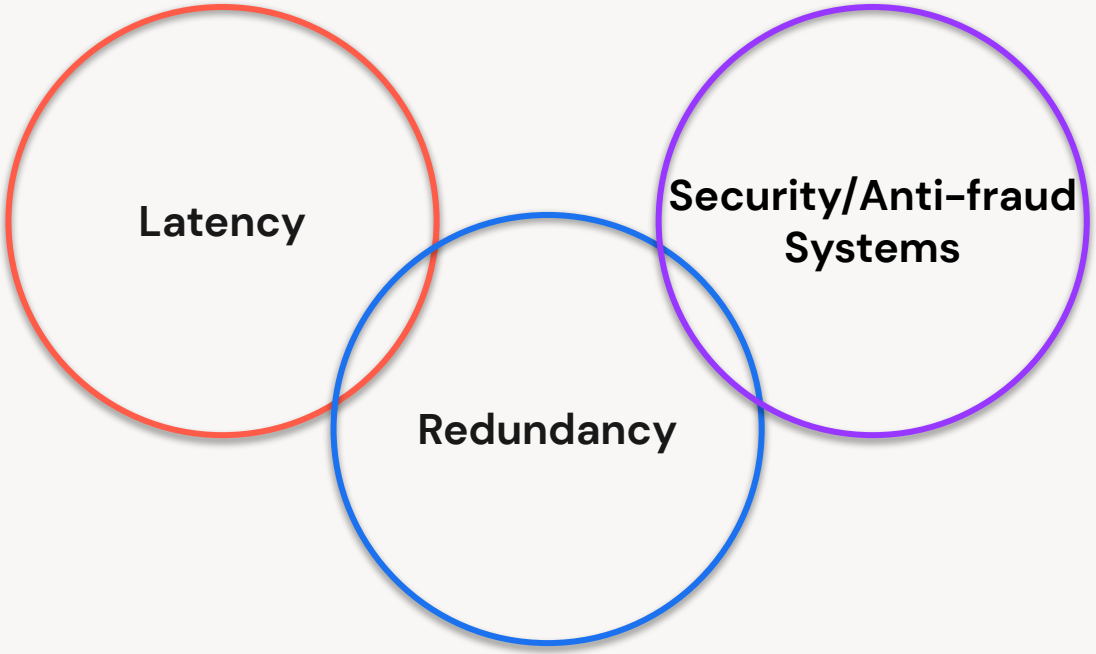
Normal operations vs on-sales

Tradeoffs
might **change**



Normal operations vs on-sales

Tradeoffs
might **change**



Normal operations vs on-sales

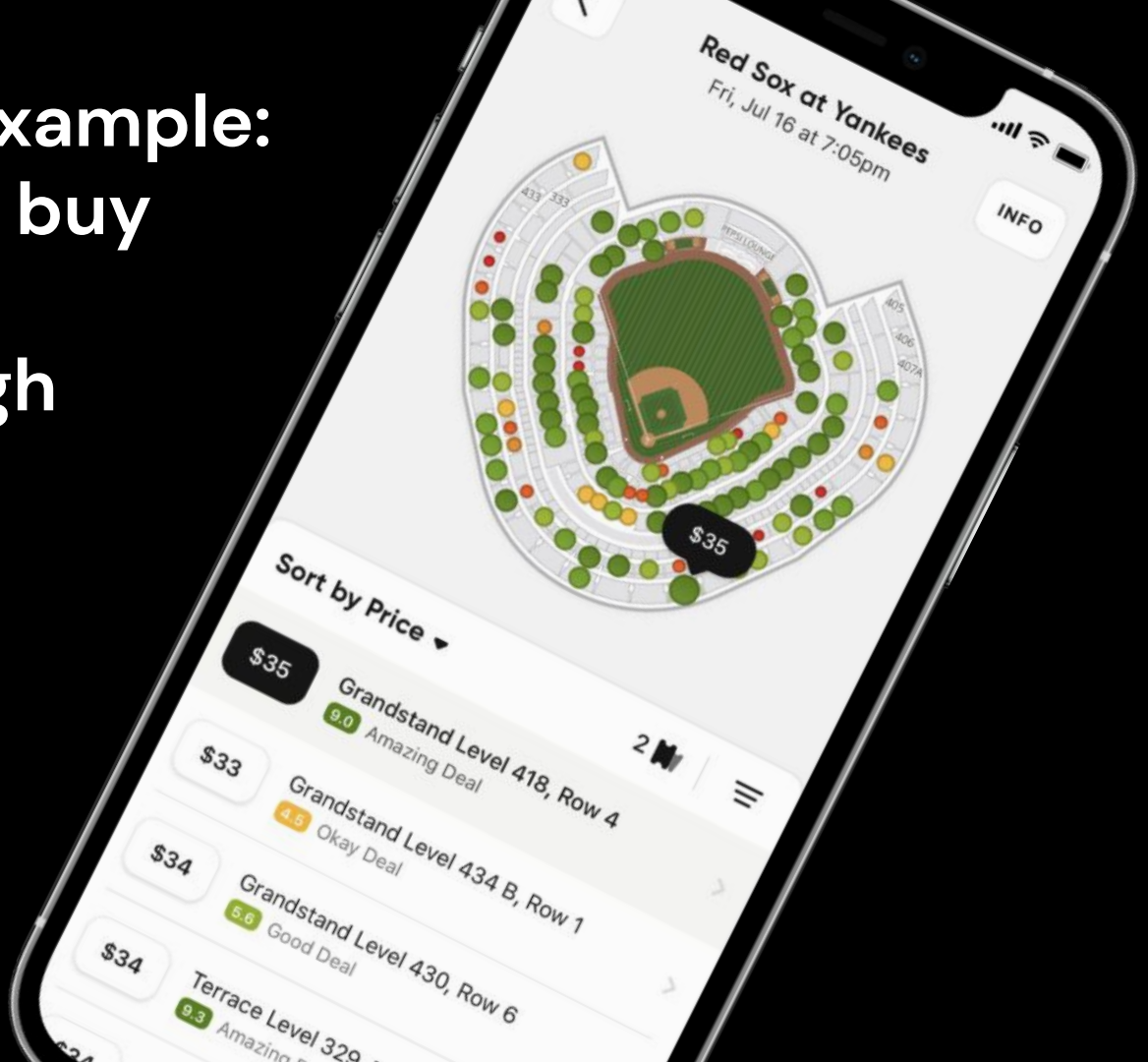
SEAT GEEK

You need to **design each mode** of operation differently.

Virtual Waiting Room

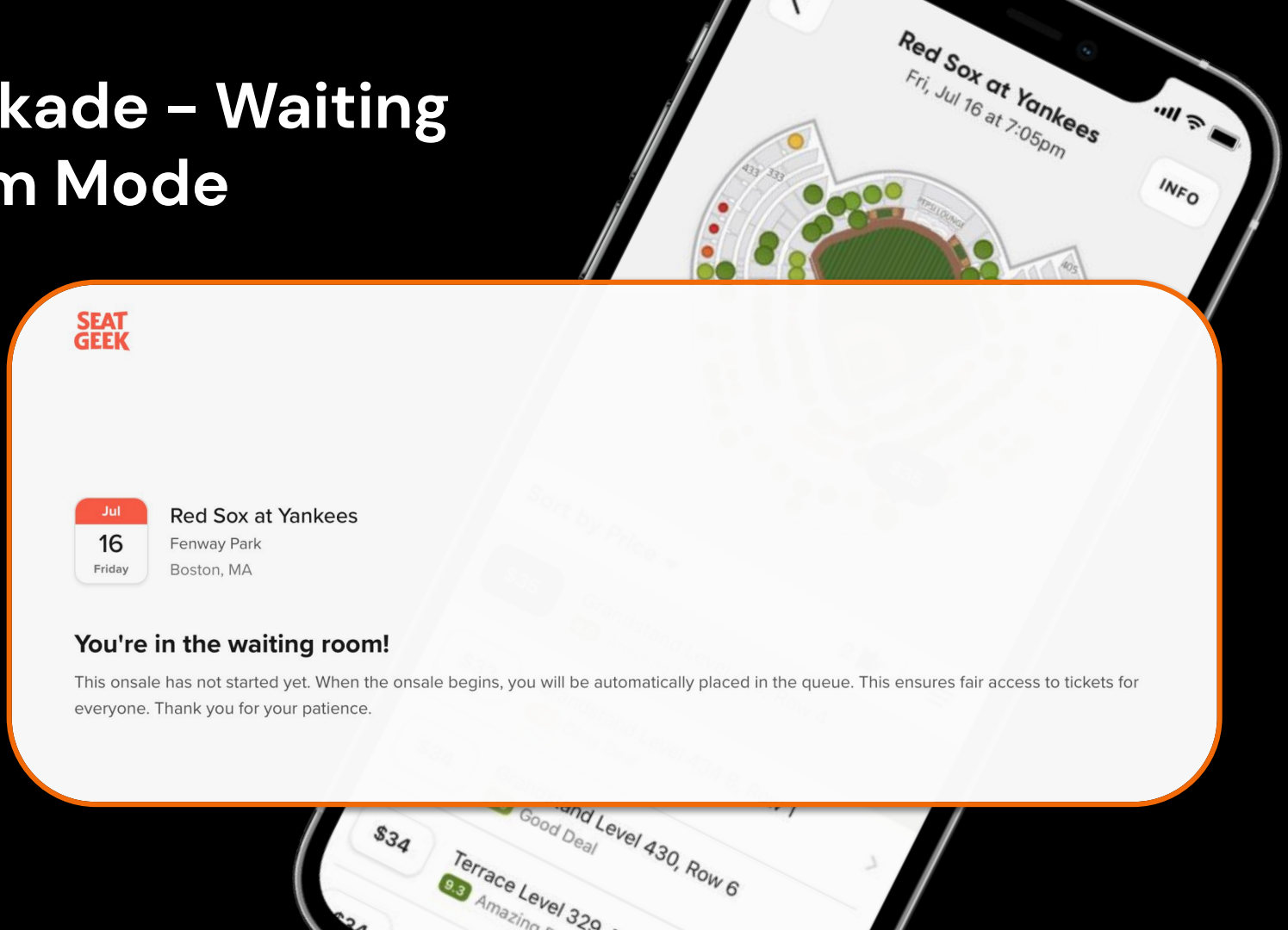
aka **Vroom**, SeatGeek in-house queueing system

Let’s see an example: I would like to buy tickets for an event with high demand...

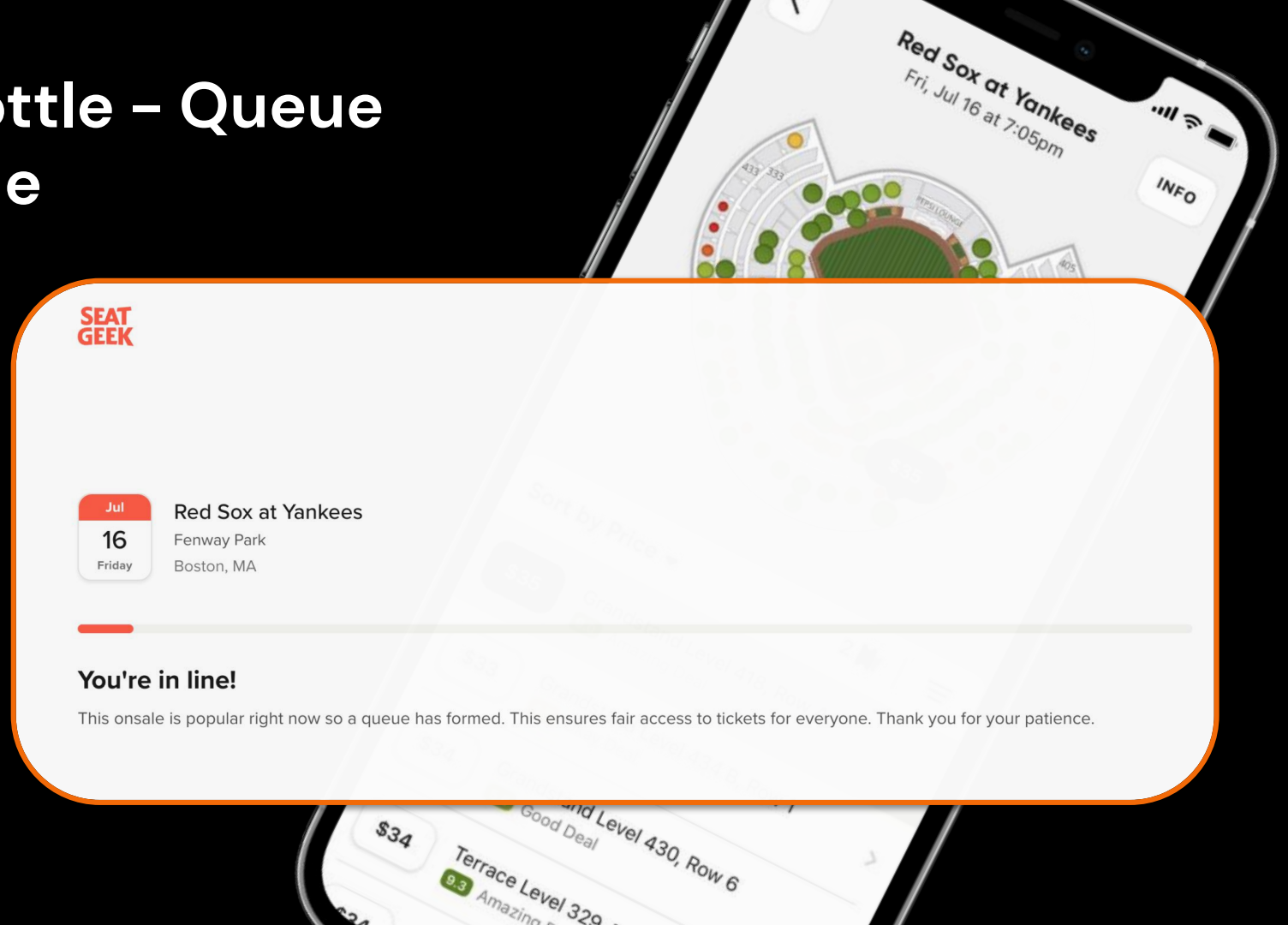


SEAT GEEK

Blockade – Waiting Room Mode



Throttle – Queue Mode




9,998,383,750,000
SEAT
GEEK

Why Do We Need a Queuing System?

SEAT GEEK

Fairness

Offers to users a fair way for purchasing (access protected resources). The fairest approach for online ticketing applications is First-In, First-Out.

Operation

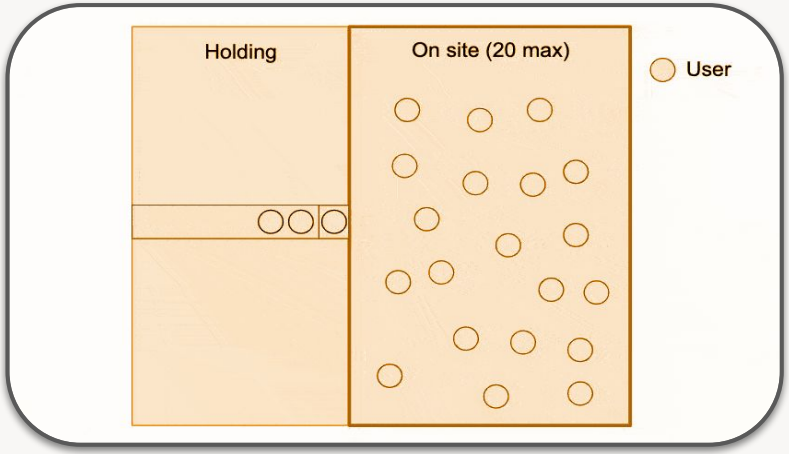
Control the amount of users accessing the available tickets. The key idea is maximize the number of sales using reserve–and-purchase strategy.

Avoid Disruptions

Minimize the risk of disruptions due to high traffic. The mechanism helps our systems to smoothly scale-up when there is unexpected high traffic.

Virtual Waiting Room Mission

Absorbs a **high traffic** and then pipes it to a **constant traffic** in our infrastructure.



Considerations when building a queuing system

SEAT GEEK

Considerations when building a queuing system

SEAT GEEK

Stateless

A stateless version of waiting room is a known pattern for CDN providers. It is similar to rate limiting, there is no ordering guarantees

Considerations when building a queuing system

SEAT GEEK

Stateless

A stateless version of waiting room is a known pattern for CDN providers. It is similar to rate limiting, there is no ordering guarantees

Stateful

When people must have a numbered position in the queue, we need to manage the state of the queue using backend service

Considerations when building a queuing system

SEAT GEEK

Stateless

A stateless version of waiting room is a known pattern for CDN providers. It is similar to rate limiting, there is no ordering guarantees

Stateful

When people must have a numbered position in the queue, we need to manage the state of the queue using backend service

Random Selection

Based on maximum threshold for the number of concurrent visitors allowed, random users are selected to get in the protected zone

Considerations when building a queuing system

SEAT GEEK

Stateless

A stateless version of waiting room is a known pattern for CDN providers. It is similar to rate limiting, there is no ordering guarantees

Stateful

When people must have a numbered position in the queue, we need to manage the state of the queue using backend service

Random Selection

Based on maximum threshold for the number of concurrent visitors allowed, random users are selected to get in the protected zone

First-In First-Out

Linear model that lines up and admits users based on the order in which their requests were received. It is the fairest model

Considerations when building a queuing system

SEAT GEEK

Stateless

A stateless version of waiting room is a known pattern for CDN providers. It is similar to rate limiting, there is no ordering guarantees

Stateful

When people must have a numbered position in the queue, we need to manage the state of the queue using backend service

Random Selection

Based on maximum threshold for the number of concurrent visitors allowed, random users are selected to get in the protected zone

First-In First-Out

Linear model that lines up and admits users based on the order in which their requests were received. It is the fairest model

Queue Management

Operators manage the queue during on sales, data like exit rate and pause/release the queue draining are constantly updated

Considerations when building a queuing system

SEAT GEEK

Stateless

A stateless version of waiting room is a known pattern for CDN providers. It is similar to rate limiting, there is no ordering guarantees

Stateful

When people must have a numbered position in the queue, we need to manage the state of the queue using backend service

Random Selection

Based on maximum threshold for the number of concurrent visitors allowed, random users are selected to get in the protected zone

First-In First-Out

Linear model that lines up and admits users based on the order in which their requests were received. It is the fairest model

Queue Management

Operators manage the queue during on sales, data like exit rate and pause/release the queue draining are constantly updated

Metrics

What is the size of audience waiting for on sale start? How long an user has waited in the queue? We measure everything.

Stateless

or

Stateful?

SEAT GEEK

Stateless

or

Stateful?

```
const originHitRate = 0.3
if (Math.random() <= originHitRate) {
  console.log("A lucky user goes to the origin")
  return true
}
```

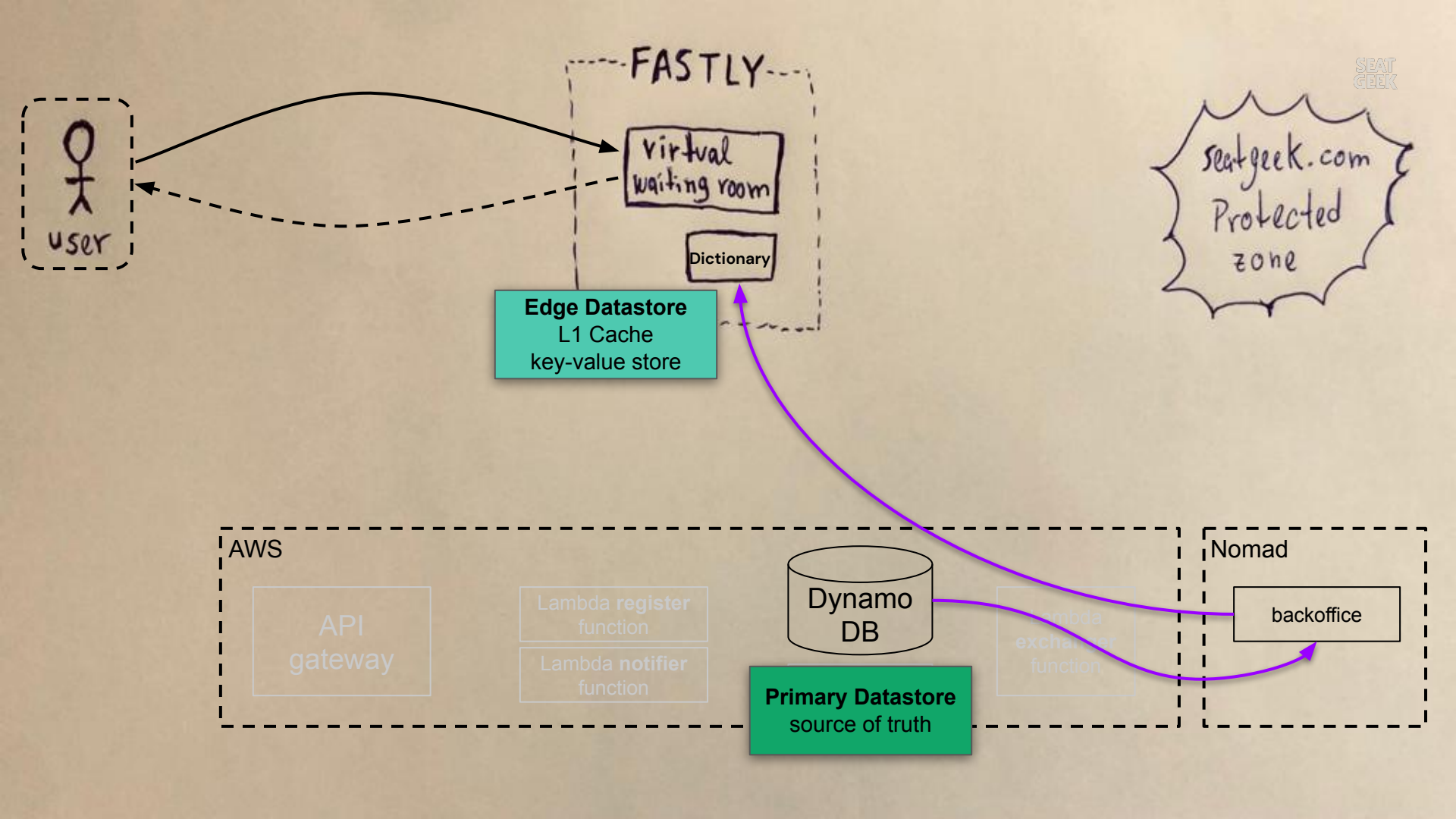
Hybrid Model: Stateless and Stateful Combined

SEAT GEEK

We chose both, the red and the blue pills

- Due the **First-In First-Out** requirement, we manage the queue state in the backend (AWS Lambda and DynamoDB).
- Validation and Routing logic run in the Edge (Fastly CDN). It minimizes the request round trip, reduces latency and costs.

Because of this hybrid approach, we need to synchronize the state between those two models.

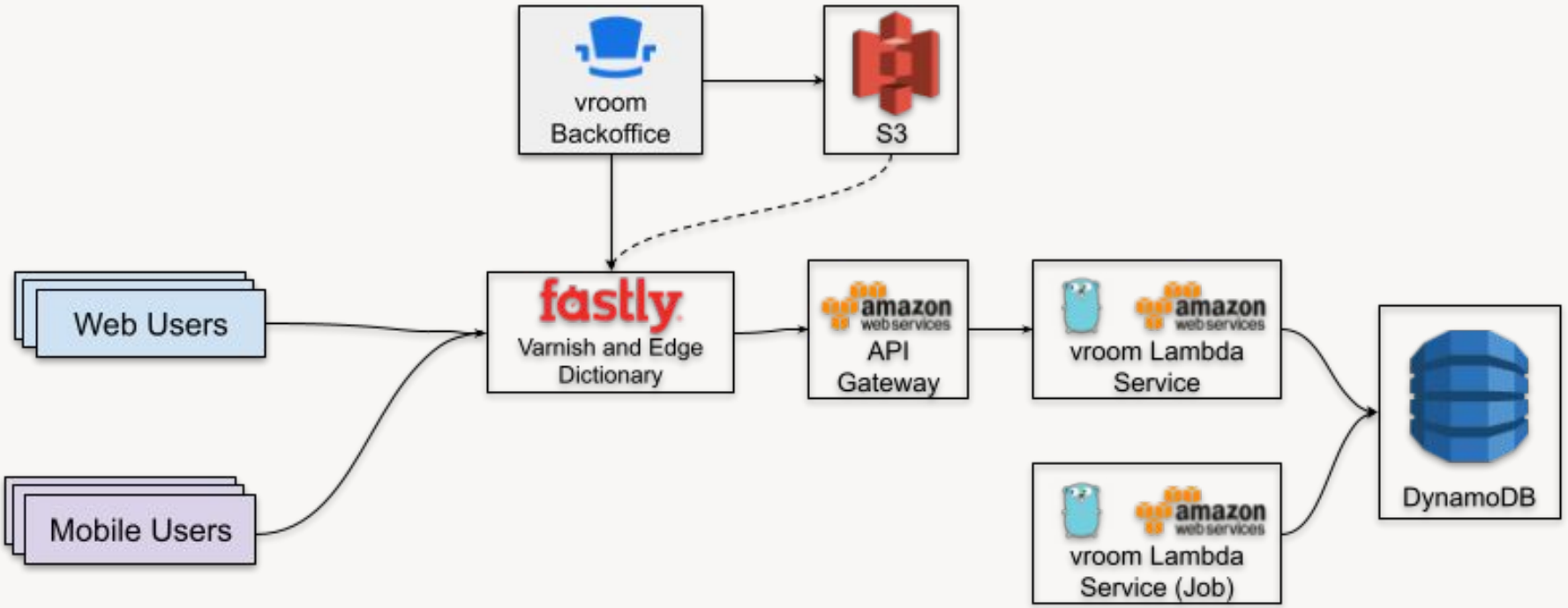

FASTLY
virtual waiting room
Dictionary
user
Edge Datastore
L1 Cache
key-value store
seatgeek.com
Protected
zone
AWS
API
gateway
Lambda register
function
Lambda notifier
function
Dynamo
DB
Primary Datastore
source of truth
Nomad
backoffice

Virtual Waiting Room Tech Stack

SEAT GEEK

Frontend	CDN	Backend	Storage	Observability
				
	Fastly VCL			
	Fastly Edge Dictionaries			

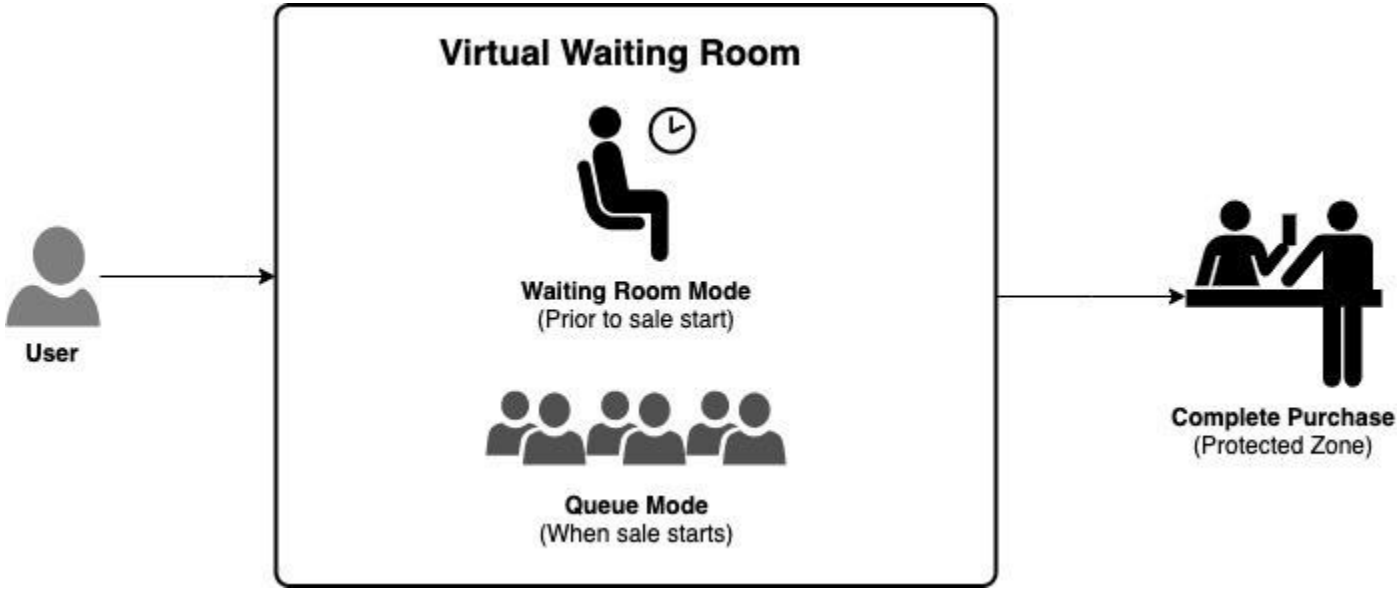
Virtual Waiting Room Stack Overview



How Does it Work?

SEAT GEEK

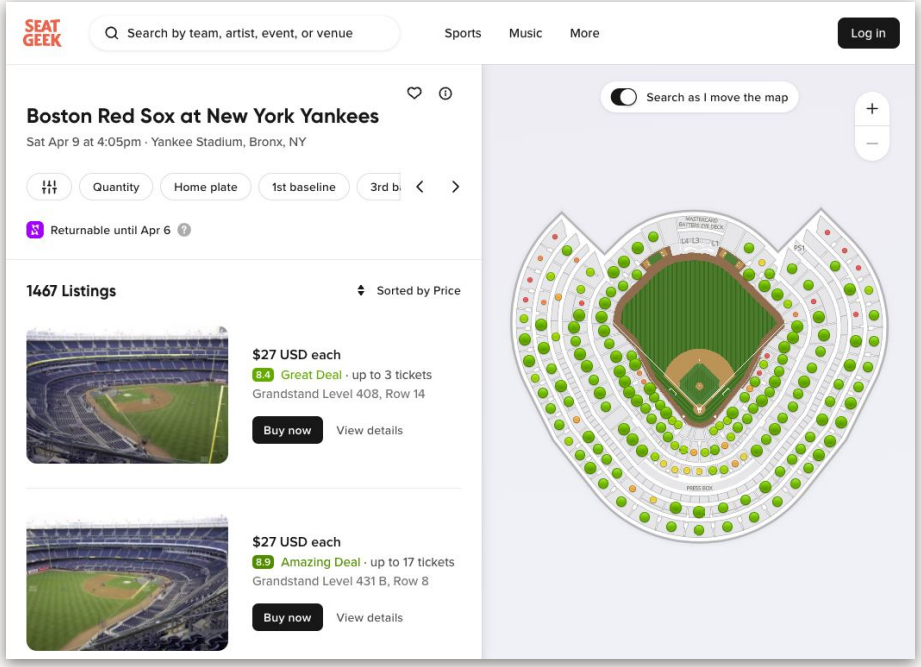
Virtual Waiting Room Operation Modes



What is a Protected Zone?

https://seatgeek.com/red-sox-at-yankees-tickets/4-9-2022-bronx-new-york-yankee-stadium/mlb/r/5465018

Protected Zone: red-sox-at-yankees

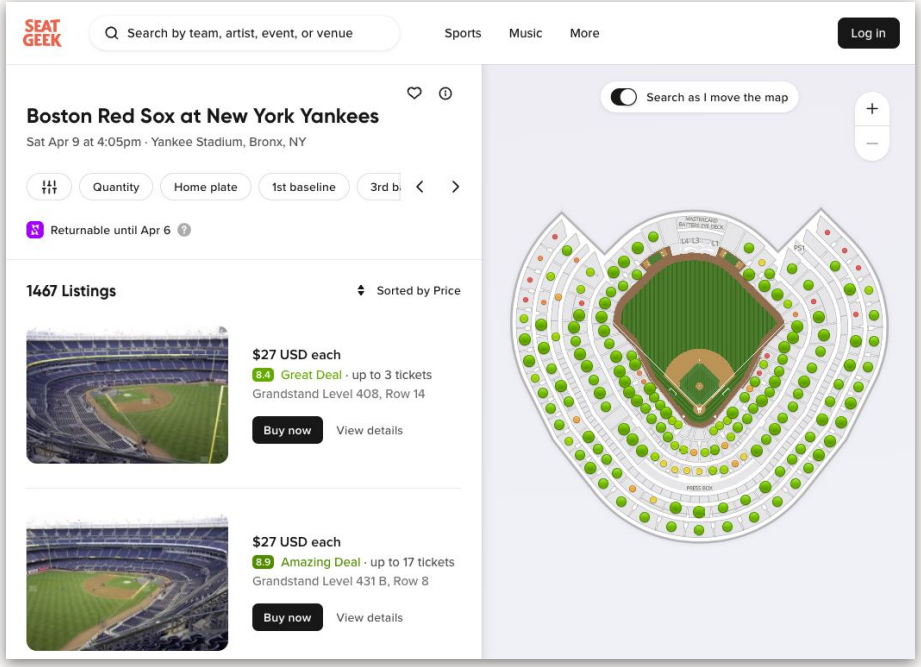


What is a Protected Zone?

https://seatgeek.com/red-sox-at-yankees-tickets/4-9-2022-bronx-new-york-yankee-stadium/mlb/r/5465018

```go
// swagger:enum ProtectedZoneState
type State string

const (
    StateDraft      State = "draft"
    StateBlockade   State = "blockade"
    StateThrottle   State = "throttle"
    StateDone       State = "done"
)

// swagger:model ProtectedZone
 type ProtectedZone struct {
    UUID          string          `json:"uuid"`
    Name          string          `json:"name"`
    Resources     []Resource      `json:"resources"`
    EventDetails *EventDetails   `json:"event_details,omitempty"`
    State         State           `json:"state"`
    Limits        Limits          `json:"limits"`
    Timeline      *Timeline       `json:"timeline,omitempty"`
}
```

Protected Zone: red-sox-at-yankees

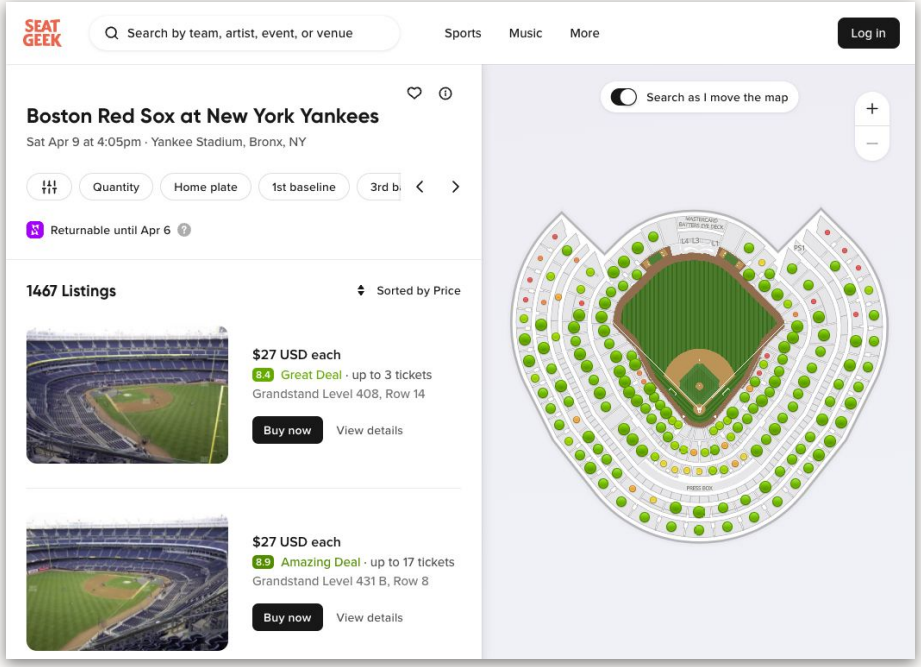


What is a Protected Zone?

https://seatgeek.com/red-sox-at-yankees-tickets/4-9-2022-bronx-new-york-yankee-stadium/mlb/r/5465018

Only requests with **Access Tokens** are routed to Protected Zones

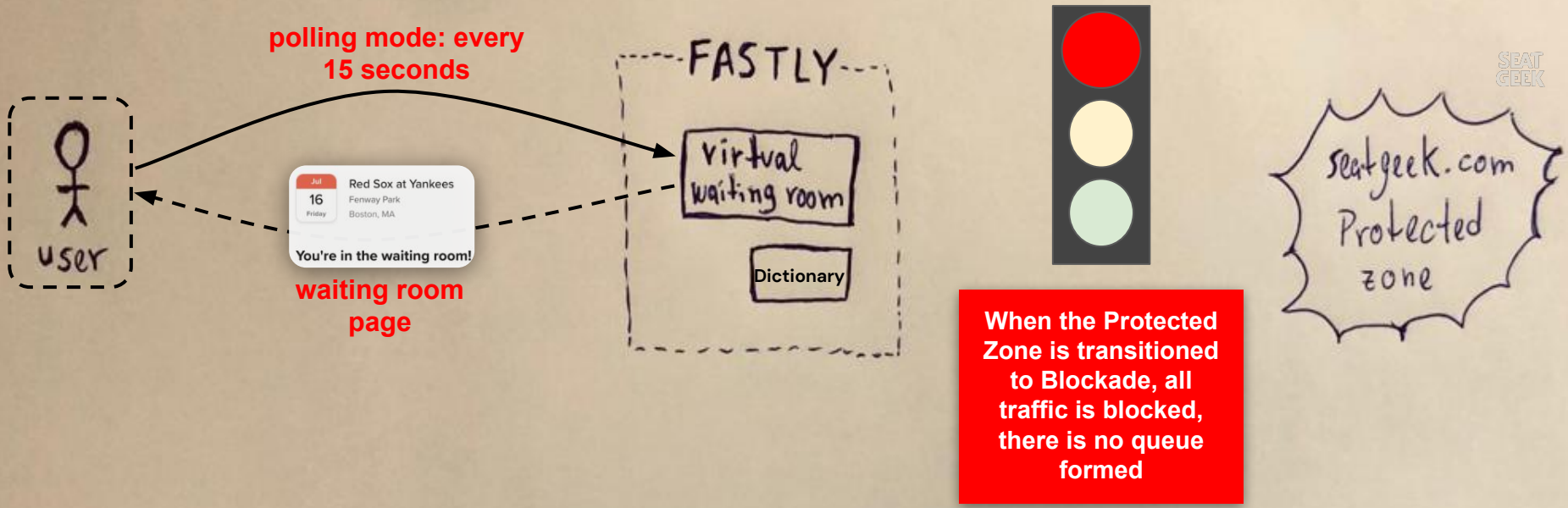
Protected Zone: red-sox-at-yankees



The Virtual Waiting Room Main States

Transitions from Blockade to Throttle State.

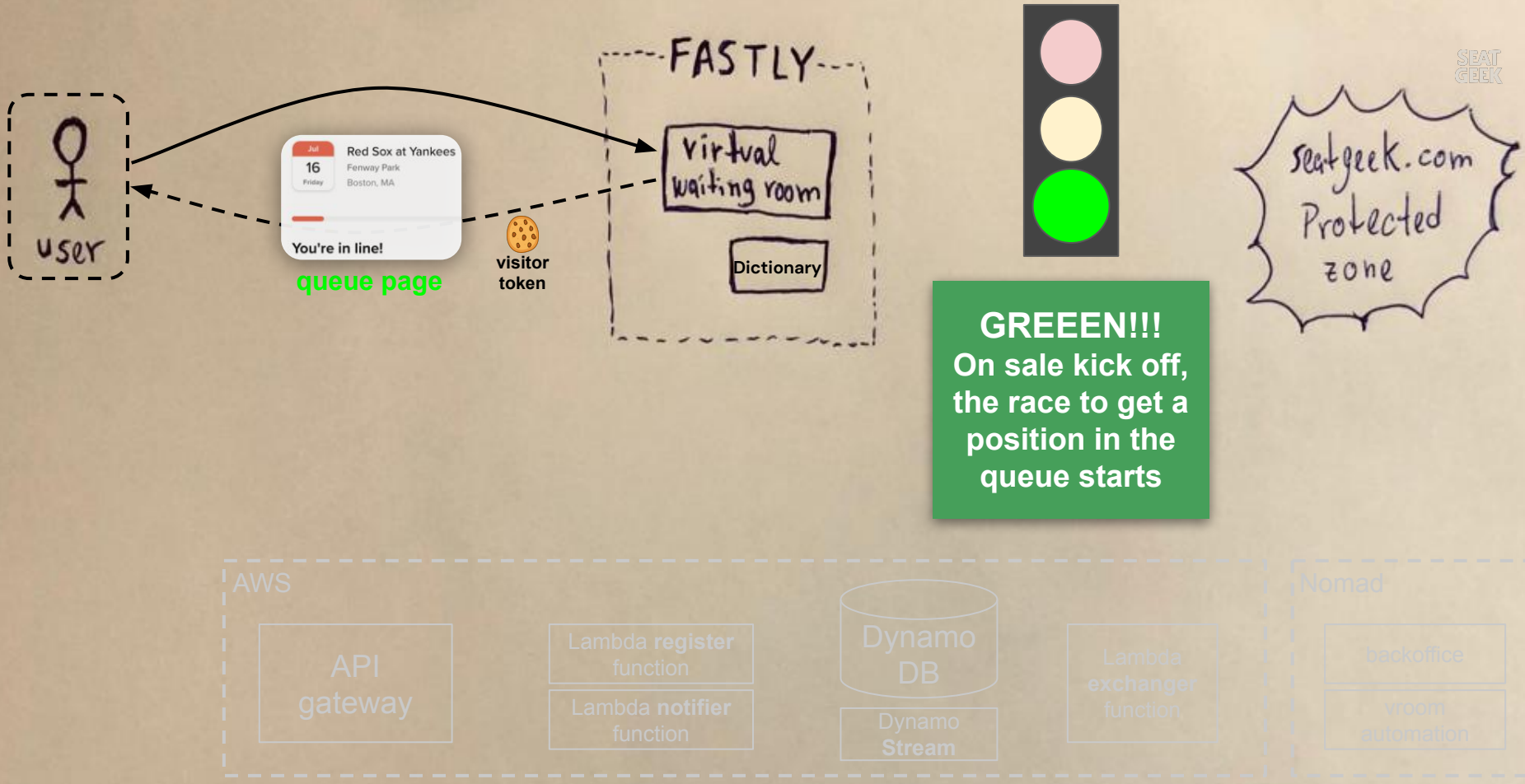
All traffic to the protected zone is blocked, showing a waiting room page to end-users.

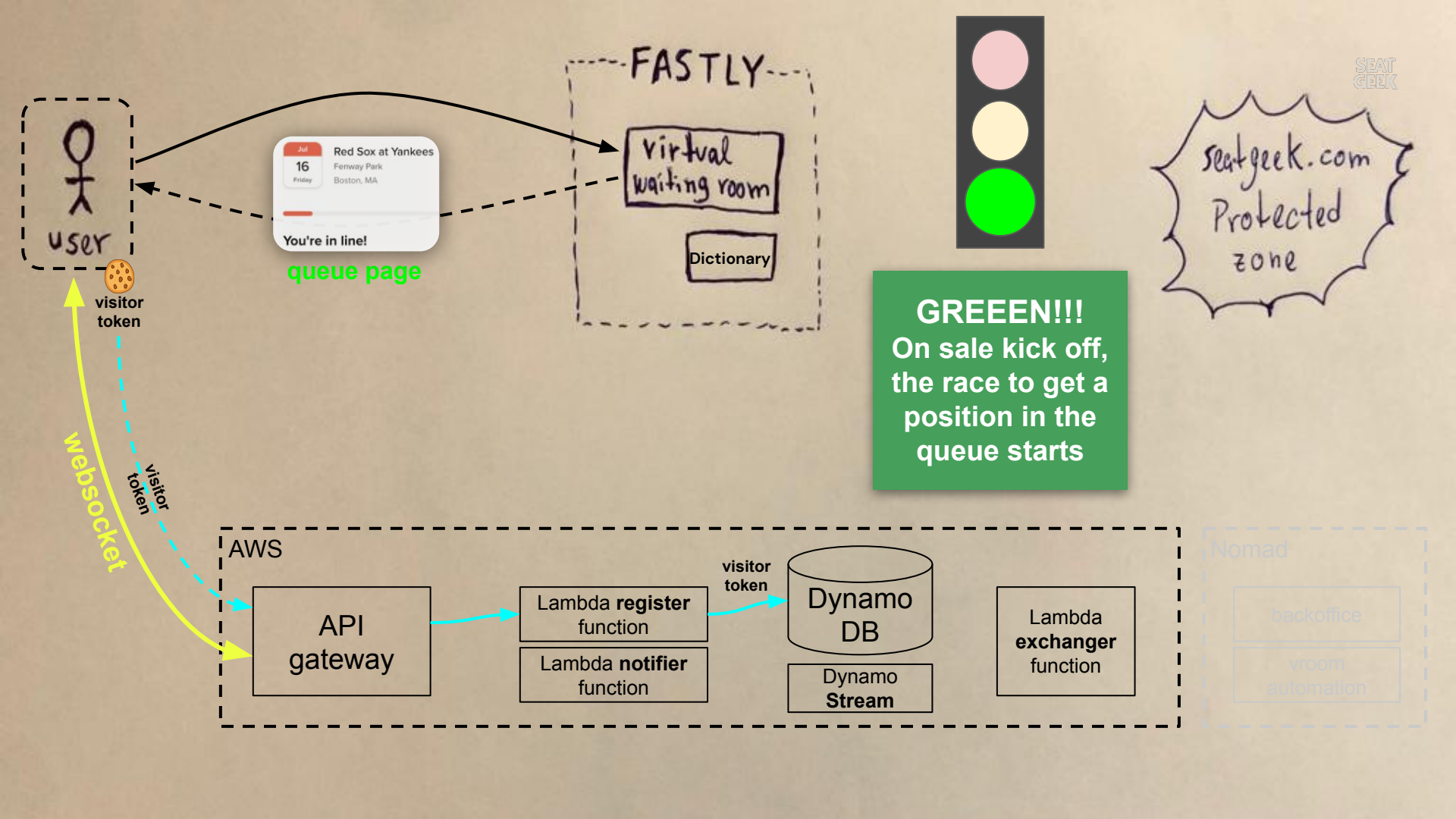

polling mode: every 15 seconds
FASTLY
virtual waiting room
Dictionary
user
Jul
16
Friday
Red Sox at Yankees
Fenway Park
Boston, MA
You're in the waiting room!
waiting room page
seatgeek.com Protected zone
When the Protected Zone is transitioned to Blockade, all traffic is blocked, there is no queue formed
AWS
API gateway
Lambda register function
Lambda notifier function
Dynamo DB
Dynamo Stream
Lambda exchanger function
Nomad
backoffice
vroom automation

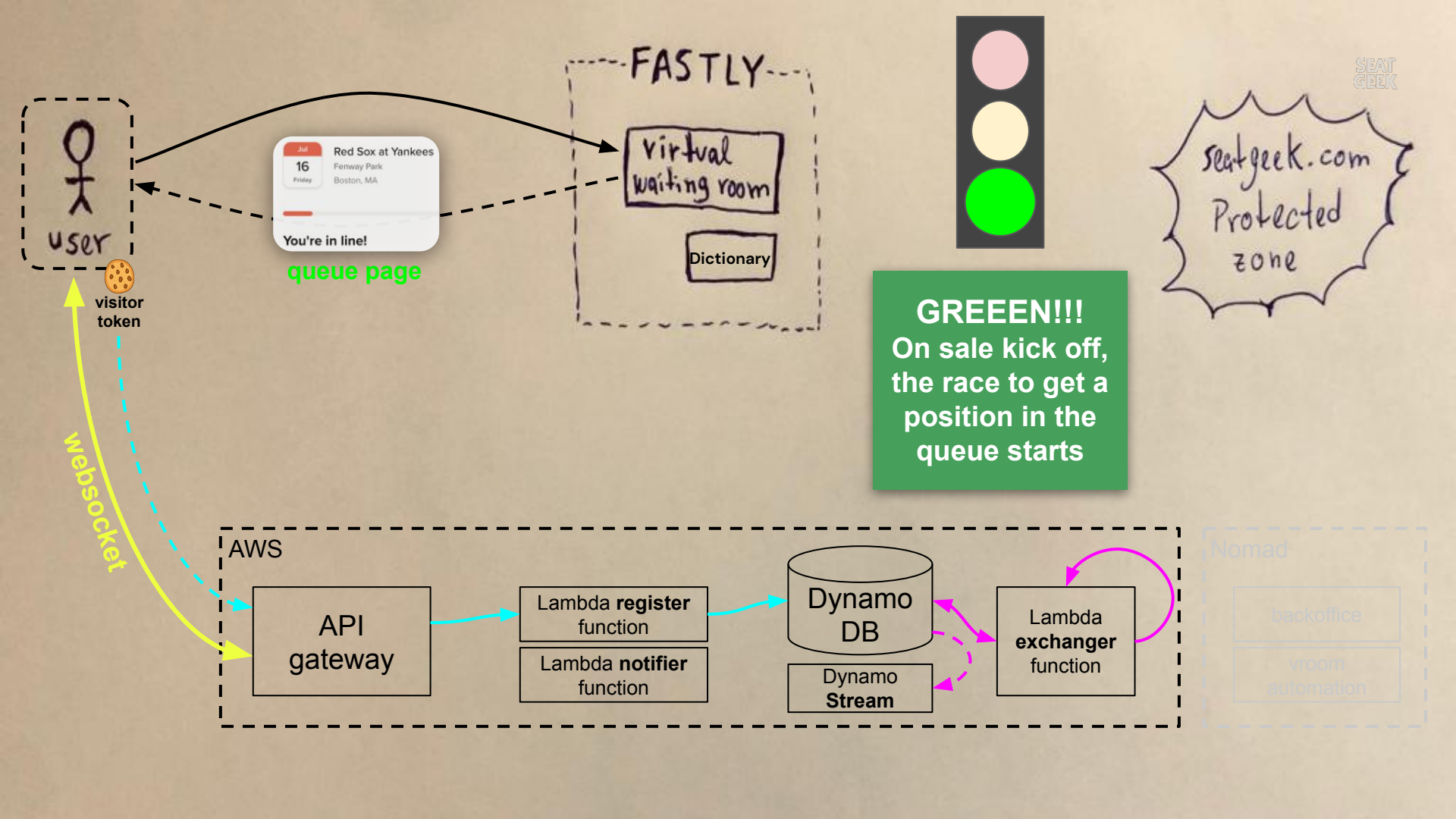
The Virtual Waiting Room Main States

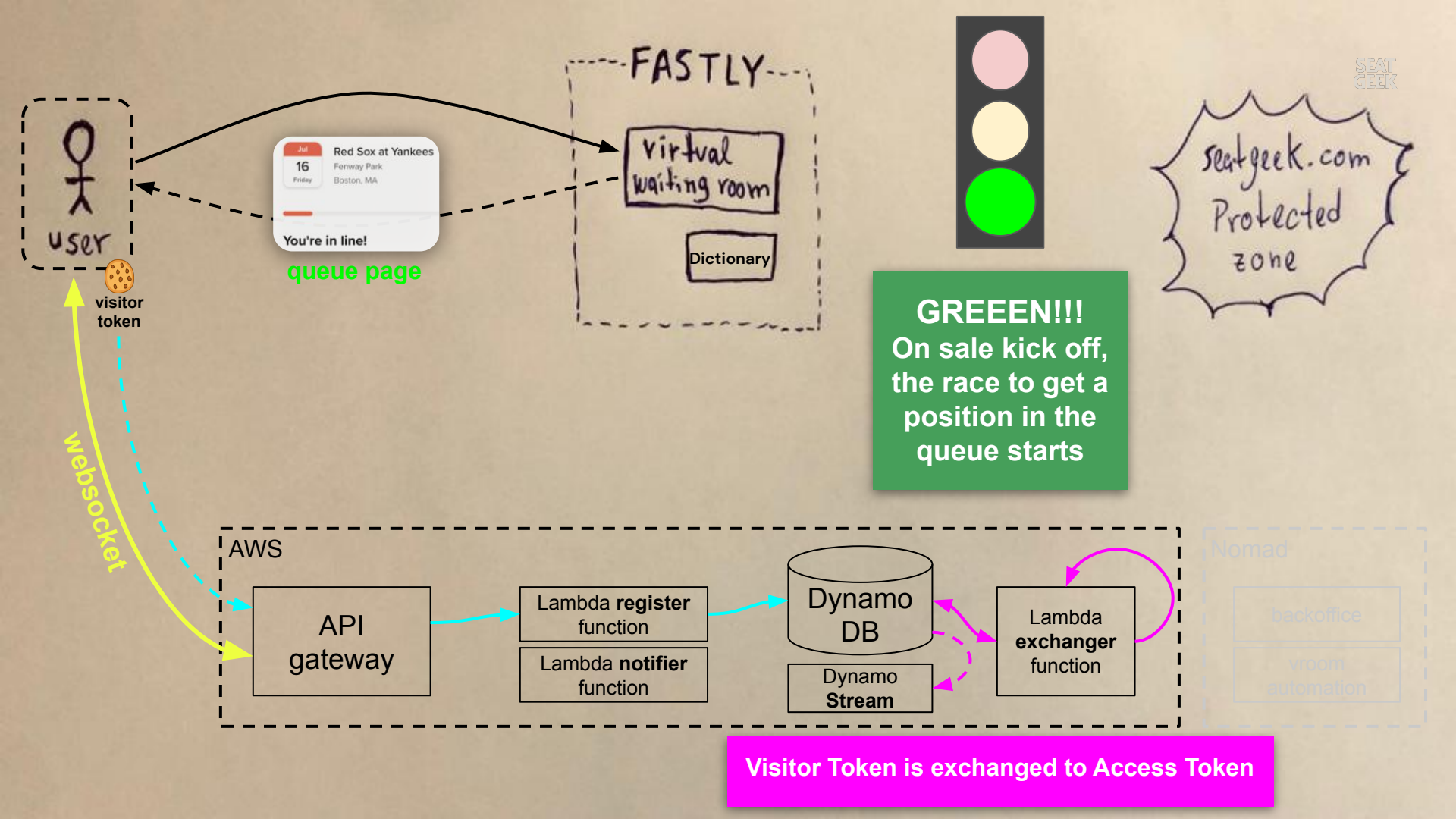
Transitions from Blockade **to Throttle State.**

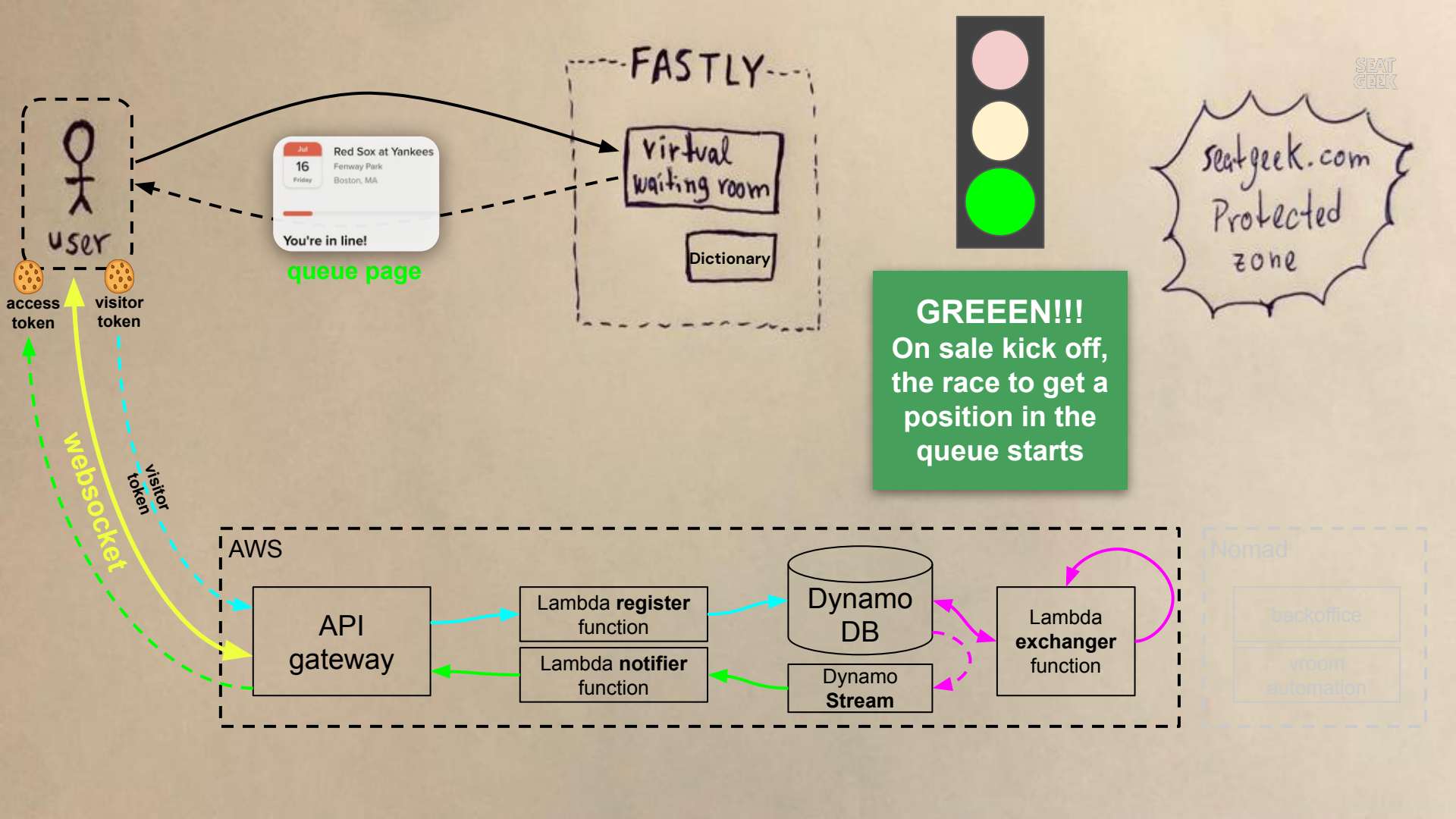
Allows a configurable number of end-users per minute into the protected zone. After initial seeding, users are allowed in First-In, First-Out (FIFO).

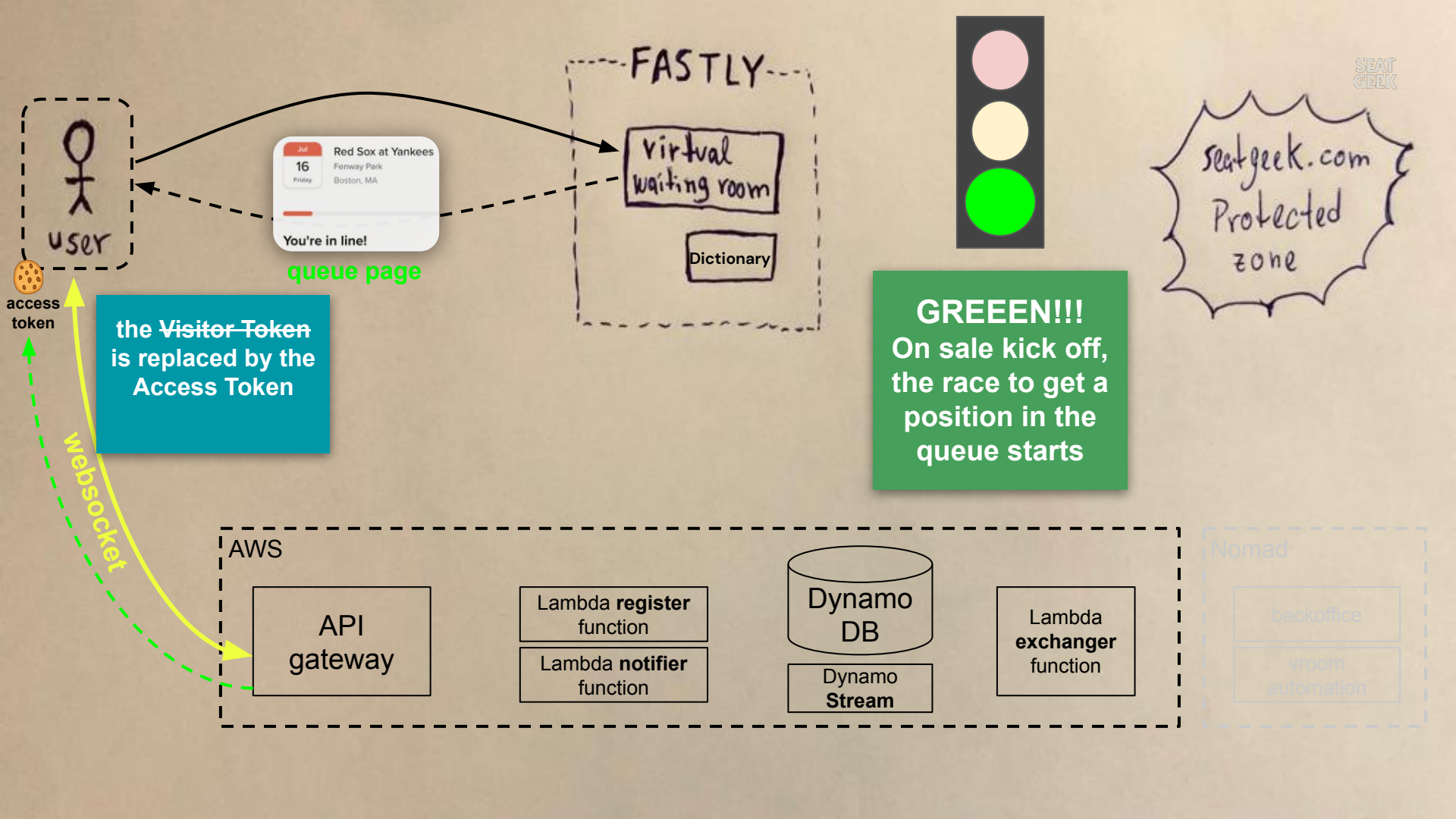

FASTLY
virtual waiting room
Dictionary
user
Jul
16
Friday
Red Sox at Yankees
Fenway Park
Boston, MA
You're in line!
queue page
visitor token
GREEEN!!!
On sale kick off, the race to get a position in the queue starts
seatgeek.com Protected zone
SEAT GEEK
AWS
API gateway
Lambda register function
Lambda notifier function
Dynamo DB
Dynamo Stream
Lambda exchanger function
Nomad
backoffice
vroom automation

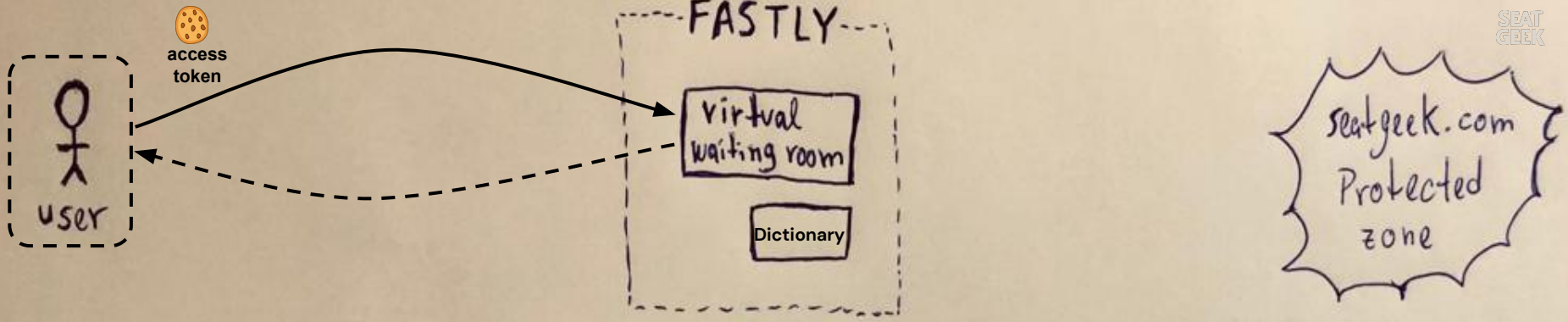

FASTLY
virtual waiting room
Dictionary
user
visitor token
Jul
16
Friday
Red Sox at Yankees
Fenway Park
Boston, MA
You're in line!
queue page
seatgeek.com Protected zone
GREEEN!!!
On sale kick off, the race to get a position in the queue starts
websocket
visitor token
AWS
API gateway
Lambda register function
Lambda notifier function
visitor token
Dynamo DB
Dynamo Stream
Lambda exchanger function
Nomad
backoffice
vroom automation


FASTLY
virtual waiting room
Dictionary
user
visitor token
Jul
16
Friday
Red Sox at Yankees
Fenway Park
Boston, MA
You're in line!
queue page
seatgeek.com Protected zone
GREEEN!!!
On sale kick off, the race to get a position in the queue starts
websocket
AWS
API gateway
Lambda register function
Lambda notifier function
Dynamo DB
Dynamo Stream
Lambda exchanger function
Nomad
backoffice
vroom automation


FASTLY
virtual waiting room
Dictionary
user
visitor token
Jul
16
Friday
Red Sox at Yankees
Fenway Park
Boston, MA
You're in line!
queue page
seatgeek.com Protected zone
GREEEN!!!
On sale kick off, the race to get a position in the queue starts
websocket
AWS
API gateway
Lambda register function
Lambda notifier function
Dynamo DB
Dynamo Stream
Lambda exchanger function
Nomad
backoffice
vroom automation
Visitor Token is exchanged to Access Token


FASTLY
virtual waiting room
Dictionary
user
access token
visitor token
Jul
16
Friday
Red Sox at Yankees
Fenway Park
Boston, MA
You're in line!
queue page
seatgeek.com Protected zone
GREEEN!!!
On sale kick off, the race to get a position in the queue starts
websocket
visitor token
AWS
API gateway
Lambda register function
Lambda notifier function
Dynamo DB
Dynamo Stream
Lambda exchanger function
Nomad
backoffice
vroom automation


FASTLY
virtual waiting room
Dictionary
user
Jul
16
Friday
Red Sox at Yankees
Fenway Park
Boston, MA
You're in line!
queue page
access token
the ~~Visitor Token~~ is replaced by the Access Token
websocket
GREEEN!!!
On sale kick off, the race to get a position in the queue starts
seatgeek.com Protected zone
SEAT GEEK
AWS
API gateway
Lambda register function
Lambda notifier function
Dynamo DB
Dynamo Stream
Lambda exchanger function
Nomad
backoffice
vroom automation



With an Access Token, User gets in the Protected Zone

AWS

API gateway

Lambda **register** function

Lambda **notifier** function

Dynamo DB

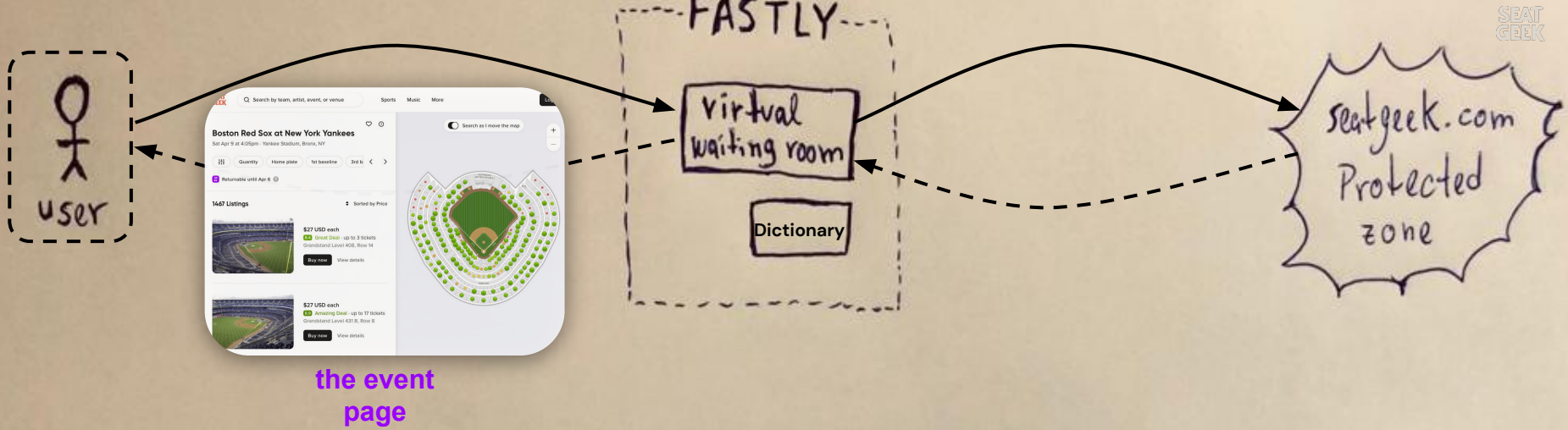
Dynamo **Stream**

Lambda **exchanger** function

Nomad

backoffice

vroom automation



the event page

There is no call to our vroom backend service

AWS

API gateway

Lambda **register** function

Lambda **notifier** function

Dynamo DB

Dynamo **Stream**

Lambda **exchanger** function

Nomad

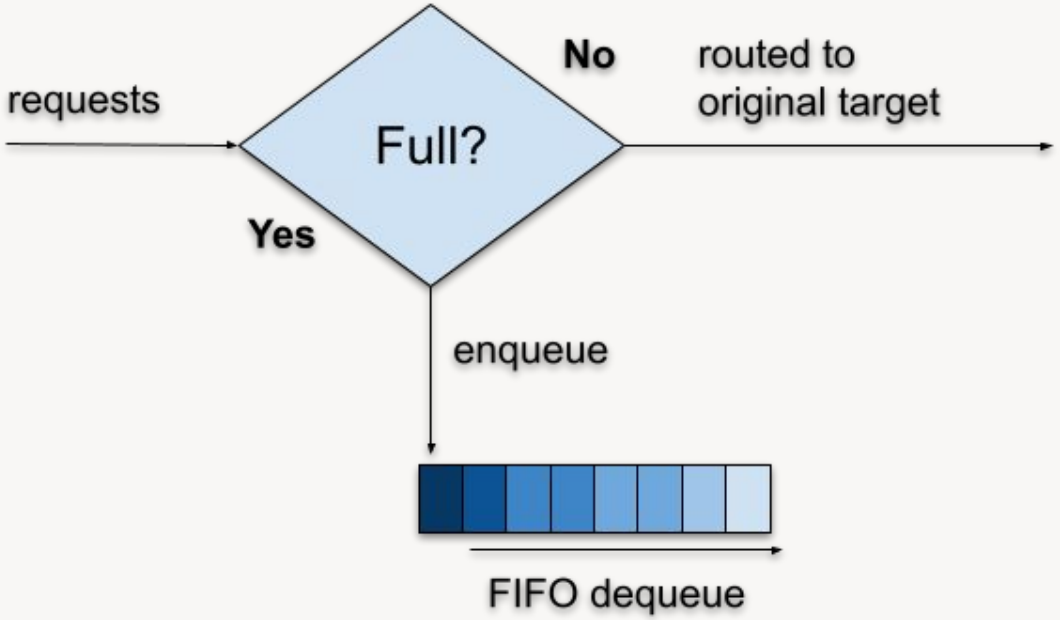
backoffice

vroom automation


BEHIND THE
SCENES

Leaky Bucket Implementation

- Does not show the queue page when bucket is empty (low traffic).
- Does not discard requests, when bucket is full (high traffic), overflow requests are routed to the queue.



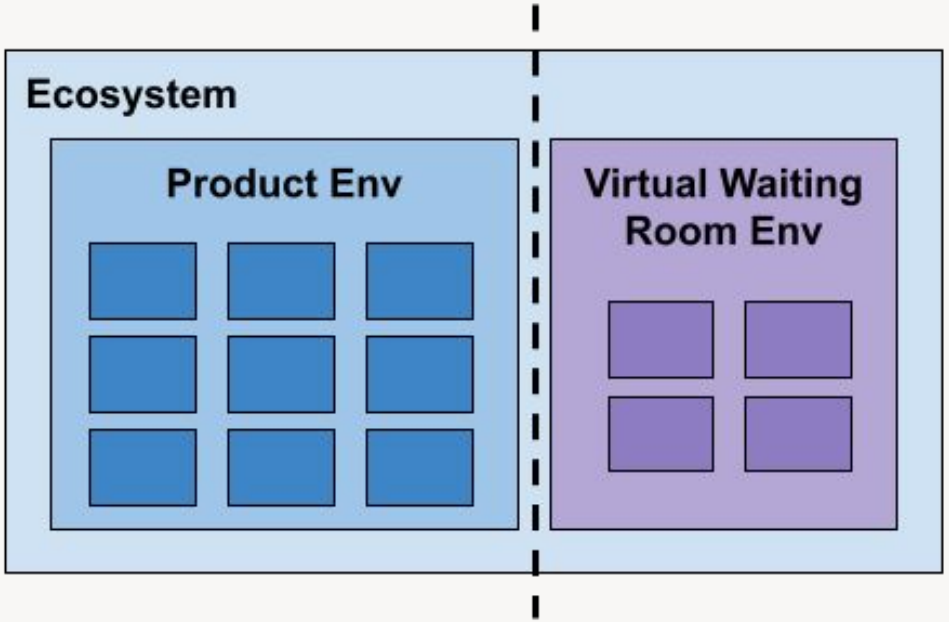
```go
func handleRequest(ctx context.Context, req events.APIGatewayProxyRequest)
(events.APIGatewayProxyResponse, error) {
    visitorToken, uuid := getVisitorTokenAndUUID(&req)
    token, pz := record(ctx, visitorToken, uuid)

    responseHeaders := map[string]string{
        // tell Fastly to not cache as this is for a single user
        "Cache-Control": "private, max-age: 0",
    }

    if err := tryExchangeVisitorToken(ctx, &token); err != nil {
        // errors for one user does not affect the bouncer capacity
        if errors.Is(err, e.ErrDynamodbStore) {
            return events.APIGatewayProxyResponse{StatusCode:
http.StatusInternalServerError, Headers: responseHeaders}, nil
        }

        // by default it returns 429 and Fastly caches it
        return events.APIGatewayProxyResponse{
            StatusCode: http.StatusTooManyRequests,
            Headers: map[string]string{
            // same as max-age but applies specifically to proxy caches (Fastly)
            "Cache-Control": fmt.Sprintf("max-age=%d, stale-if-error=60",
60-time.Now().Second()),
            },
        }, nil
    }

    responseHeaders["X-Access-Token"] = token.AccessToken

    return events.APIGatewayProxyResponse{StatusCode: http.StatusOK,
Headers: responseHeaders}, nil
}
```


```
if (req.http.X-Bouncer == "true") {
  # unset things we do not want to save with the cached object
  unset beresp.http.Set-Cookie;

  if (beresp.status == 429) {
    # cache the beresp if it is 429 response code with
    # proper Cache-Control
    set beresp.cacheable = true;
  } else {
    # turn off caching for everything else
    set beresp.cacheable = false;
  }

  return(deliver);
}
```

Why Are We Using AWS Lambda?

Premisse: Virtual Waiting Room should run aside of Product Environment. It provides isolation and avoids cascade effect in case of failures.

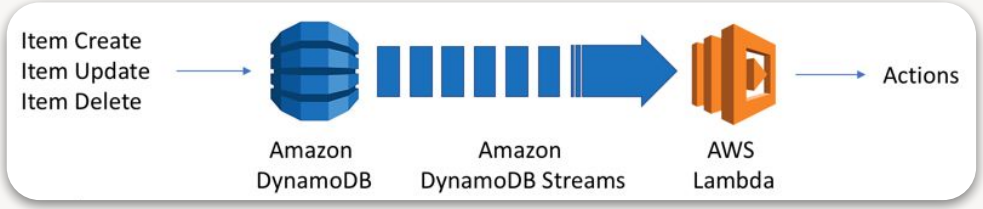
- Lambda simplifies the infrastructure management.
- Great support for concurrent executions.
- Save costs by paying only for the compute time you use.



Why Are We Using DynamoDB?

DynamoDB Streams

It is an ordered flow of information about changes to items in a DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table.



Time to Live (TTL) Attribute

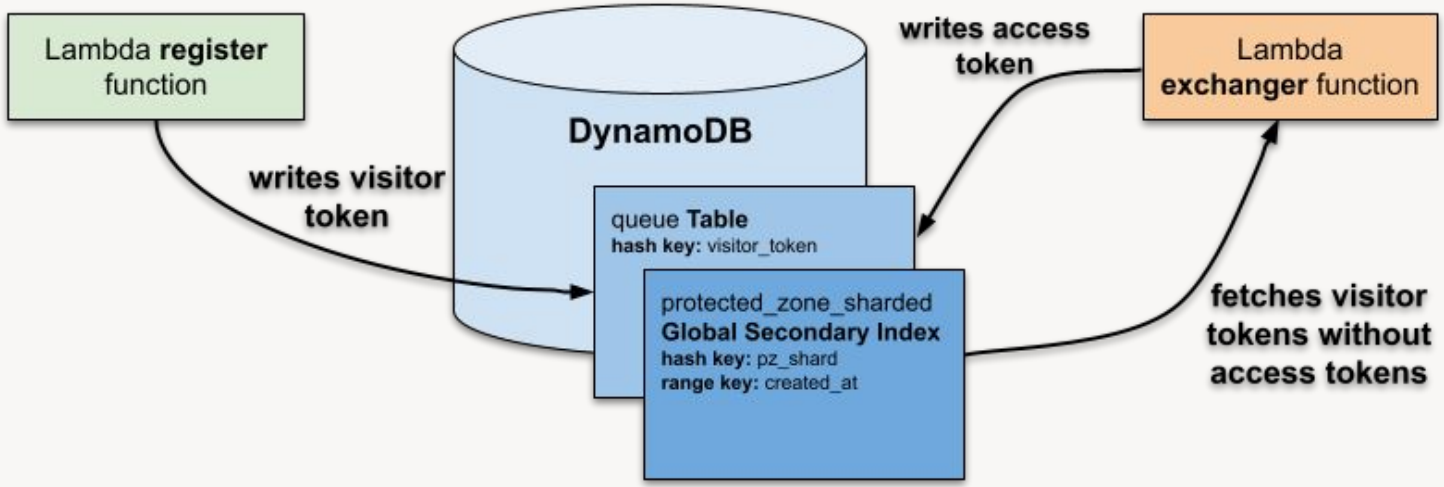
DynamoDB allows you to define a per-item timestamp to determine when an item is no longer needed. DynamoDB deletes the item from your table without consuming any write throughput (WCU).

DynamoDB Partition

Each partition in DynamoDB supports:

- 3k Read Capacity Unit (RCU) per second
- 1K Write Capacity Unit (WCU) per second

DynamoDB **throttles** if a single partition receives more requests than supported.



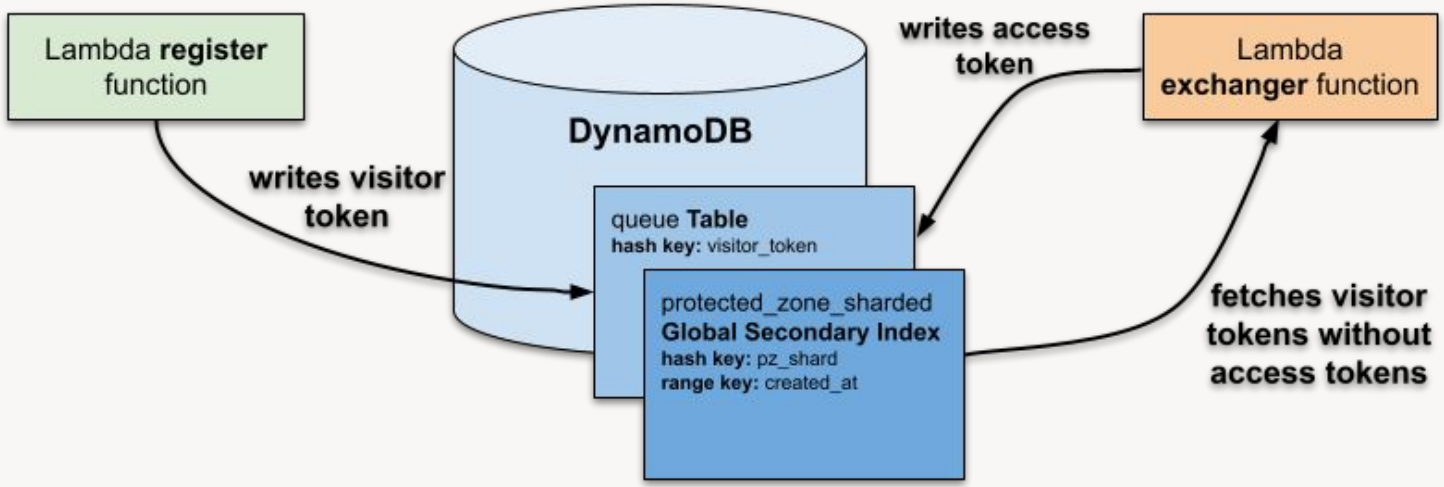
DynamoDB Partition – Sharding

Multiples the capacity by total of shards.

Let's suppose 10 shards, it will scale up:

- From 3k RCU to 30K RCU
- From 1K WCU to 10K WCU

```
const protectedZoneShards = 10
func (t *Token) Shard() string {
    hash := fnv.New32a()
    _, _ = hash.Write([]byte(t.VisitorToken))
    return fmt.Sprintf(
        "%s-%d",
        t.ProtectedZoneUUID,
        hash.Sum32()%protectedZoneShards,
    )
}
```

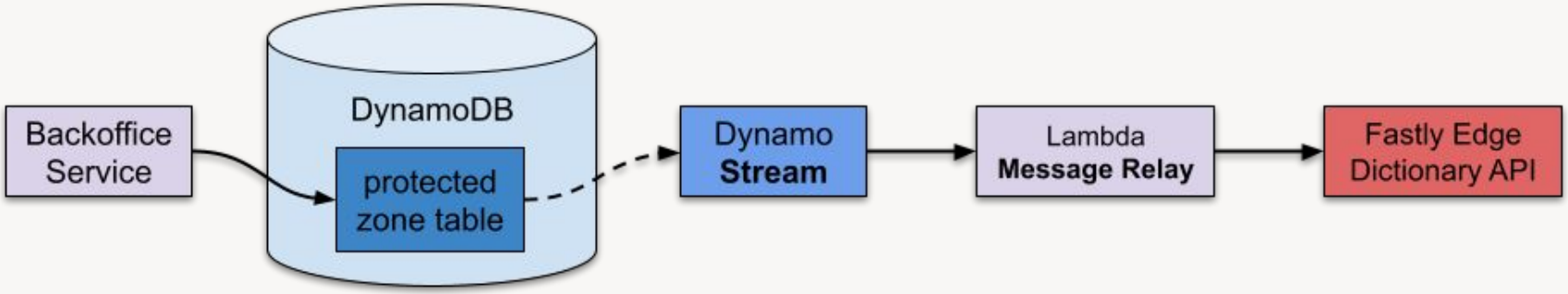


Sync Challenge: DynamoDB <-> Fastly Dictionary

SEAT GEEK

How to reliably/atomically update DynamoDB and Fastly Dictionary?

2 Phase Commit is not an option. We apply **Transactional Outbox Pattern**. It means that **DynamoDB** is the source of truth and any change (insert/update) into protected zone table is streamed using **Dynamo Stream.** It creates a separate Message Relay process that publishes the data inserted/updated into DynamoDB to Fastly Dictionary.



https://docs.fastly.com/en/guides/about-edge-dictionaries

```
table blocklist {
  "/event-1": "event_details=???&state=blockade&uuid=20b5e1cc...",
  "/event-2": "event_details=???&state=throttle&uuid=8772fde5...",
}

sub vroom_resource_metadata {
  # Do nothing if we already have the header populated
  if (std.strlen(req.http.X-Metadata) > 0) {
    return;
  }

  declare local var.metadata STRING;
  set var.metadata = table.lookup(blocklist, req.url.path);

  if (std.strlen(var.pz_metadata) > 0) {
    set req.http.X-Metadata = "/?" + urldecode(var.metadata);
    return;
  }
}

sub vroom_routing {
  declare local var.state STRING;
  set var.state = querystring.get(req.http.X-Metadata, "state");
  # Validation route logic ...
}
```

https://developer.fastly.com/solutions/examples/rust/

```
table blocklist {
  "/event-1": "event_details=???&state=blockade&uuid=20b5e1cc...",
  "/event-2": "event_details=???&state=throttle&uuid=8772fde5...",
}

#[derive(Serialize, Deserialize, Debug)]
struct Metadata {
  uuid: Option<String>,
  event_details: Option<String>,
  state: Option<String>,
}

fn vroom_resource_metadata(req: &Request) -> Option<Metadata> {
  let blocklist = Dictionary::open("blocklist");
  match blocklist.get(req.url.path) {
    Some(text) => Some(serde_json::from_str(text)?),
    _ => None,
  }
}

fn vroom_routing(req: &Request) -> Result<Response> {
  let metadata = match vroom_resource_metadata(req) {
    Some(metadata) => metadata,
    _ => return req_without_protection(req),
  };
  // validation route logic ...
}
```

Would Like to Know More Details?

AWS Architecture Blog

Build a Virtual Waiting Room with Amazon DynamoDB and AWS Lambda at SeatGeek

by Umesh Kalaspurkar, Ander Parra, Cici Xue, and Ilya Marmur | on 29 JUL 2021 | in Amazon DynamoDB, Amazon Timestream, Architecture, AWS Lambda, Intermediate (200), Retail | Permalink | Share

As retail sales, products, and customers continue to expand online, we've seen a trend t
limited quantities to larger audiences. Demand of these products can be high, due
capacity limits, or product exclusivity. Providers can then experience spikes in t
multiple event sales occur simultaneously. This increased traffic and load ca
infrastructure.

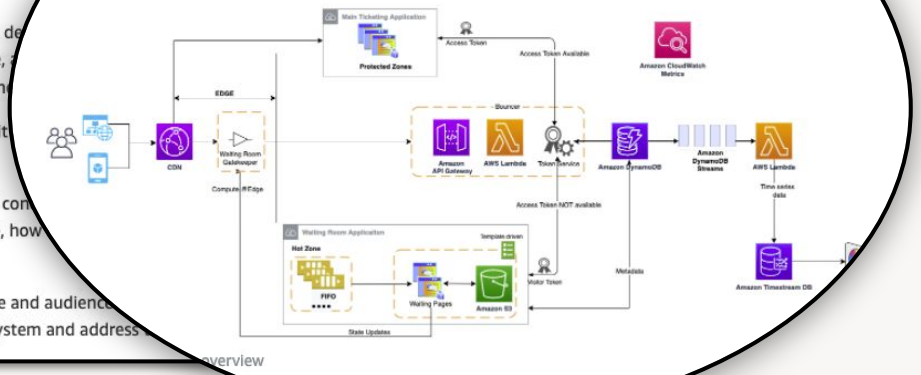
To enhance the customer experience when releasing tickets to high de
prioritization and queueing mechanism based on event type, venue,
tickets could have a different priority depending on seat type, or wh

SeatGeek previously used a third-party waiting room solution, but it

- Lack of configuration and customization capabilities
- More manual process that resulted in limiting the number of cor
- Inability to capture custom insights and metrics (for example, how
  before they dropped?)

Resolving these issues is crucial to improve the customer experience and audience
build a custom solution on AWS, in order to create a more robust system and address

the AWS services and flow that SeatGeek im
SeatGeek customer requests a protected resource like a concert ticke
resource has an active waiting room. It also confirms if the configuration rules are
tomer access. If the customer isn't allowed access to the protected resource for whatever rea
is redirected to the Virtual Waiting Room.



overview

the gate keeper service on Fastly's Compu

https://aws.amazon.com/blogs/architecture/build-a-virtual-waiting-room-with-amazon-dynamodb-and-aws-lambda-at-seatgeek/

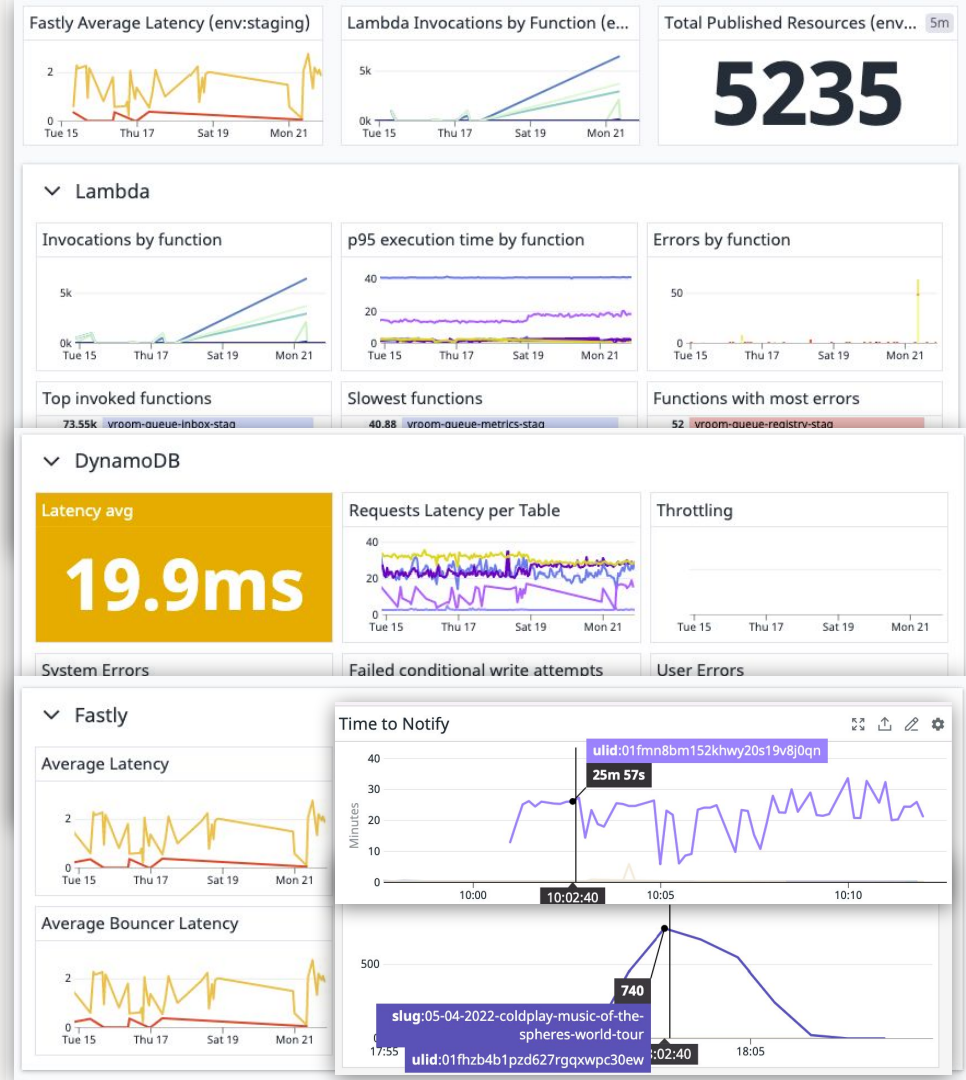
Virtual Waiting Room Observability

SEAT GEEK

DATADOG

Metrics and Alerting

- Metrics on Datadog
  - Business metrics such as time in the queue, queue length, blockade zones.
  - System metrics such as Fastly latency, DynamoDB latency, p95 execution time of lambda functions, etc.
- Metrics on AWS Timestream (Long Term Storage)
  - Available to understanding the past on sales e predict the behaviour of next ones.
- Queue Sensors
  - Stakeholders notification - via Slack - regardless queue formed



What is Next?

Next Steps for our pipeline

Automation: we are investing to reduce human interference operating our systems. Our vision is to run large on sales operated only by robots. It means, promoters design the on sale timeline and then everything else is controlled by robots. It is connected to:

- Dynamic exit rate based on traffic
- Dynamic on-call alerting severity based on critical window
- Continuous Improvement for Fraud Detection

Next Steps for our operations

Operations: We want our services to understand whether they are in "on-sale" or normal operations mode. This will inform:

- Incident response
- On-sale aware telemetry
- Service configuration
- SLOs for on-sale vs normal operations

Summary

Elasticity at all layers of your infrastructure

Virtual Waiting Rooms, or online queuing systems, are a good, simple way to control traffic surges on web applications. But they aren’t a replacement for properly scaling your infrastructure.

Leverage the toolkit you have at your disposal

DynamoDB and Lambda were a great match for us. It pays off to understand what is already built into them.

Advantages of Data Storage @ Edge

Store data at Edge (CDN) is a recent topic. There are some limitations regarding it, mainly for data manipulation and capacity. However when you have a case that fits well on it, take it!

- Content sharing and social media outlets updating large referer block lists
- Customers authenticating valid user keys at the Edge
- Publishers redirecting users to a specific country site based on geo-location
- Advertising technology companies getting the identity of users at the Edge

The End

Thank You!

Thanks to everyone for your patience and time. Hope everyone has a better understanding how SeatGeek handles high traffic during event on sales.

Questions?

If you want to know more about any of this, please reach out to one of us:

- **Anderson Parra**
  -  **@anderparra**
  -  **/anderparra**
- **Vitor Pellegrino**
  -  **@pellegrino**
  -  **/vitorpellegrino**