НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Системный дизайн современных приложений

Семинар №10-11
Как думать, что все пропало?

Монолит

Само приложение.
Что можно придумать?

1 экземпляр
Монолит
Отказ = не работает полностью

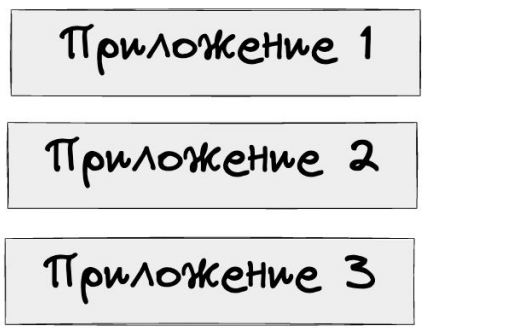
Предотвращаем отказ сервиса

Само приложение.
Что можно придумать?

1 экземпляр
Монолит
Отказ = не работает полностью

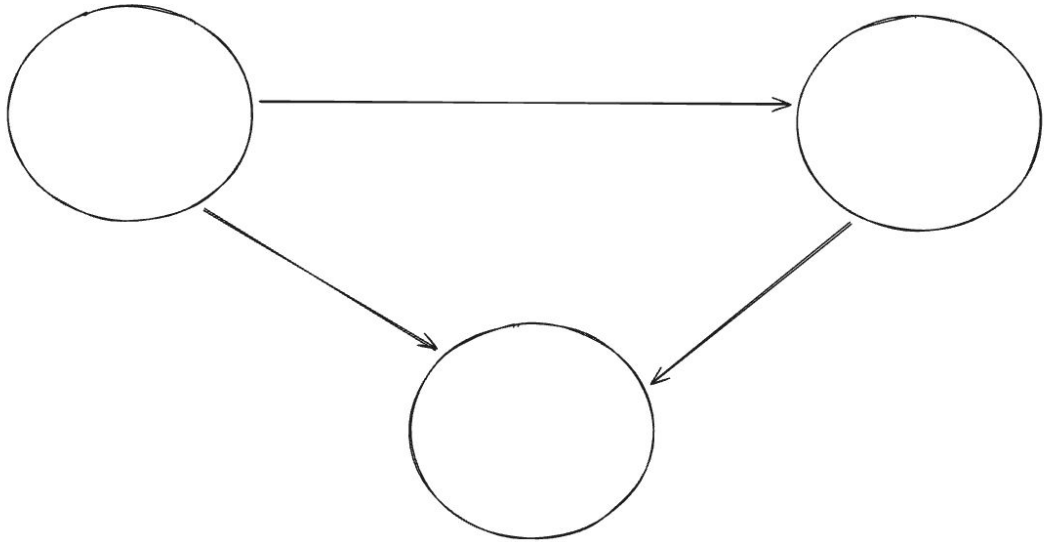
Резерв

3 экземпляра
Резервирование
Отказ одного = не работает частично



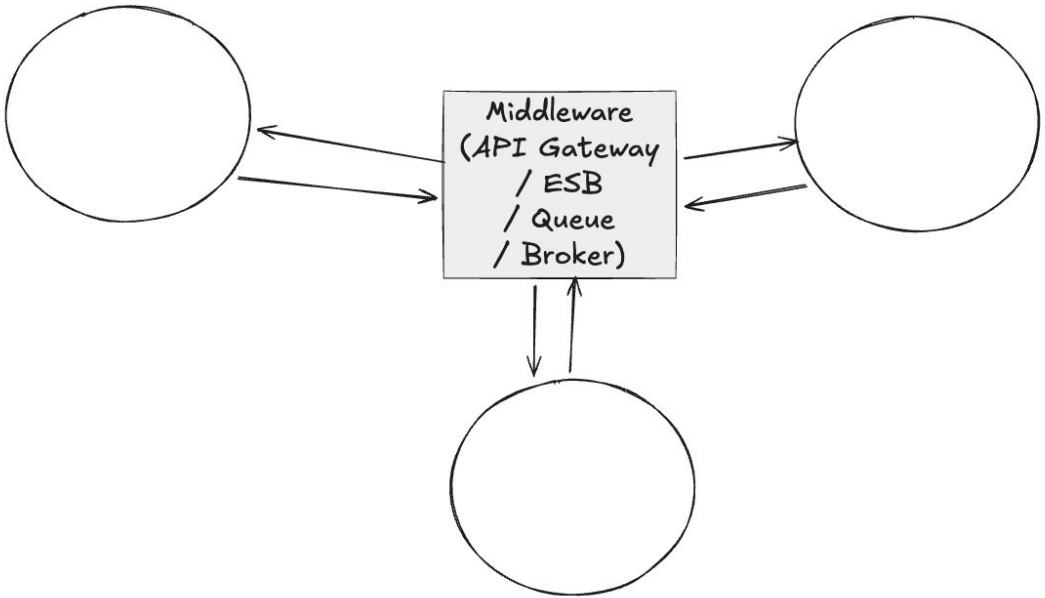
Предотвращаем отказ интеграции

Связь между сервисами (интеграции)
Что можно придумать?



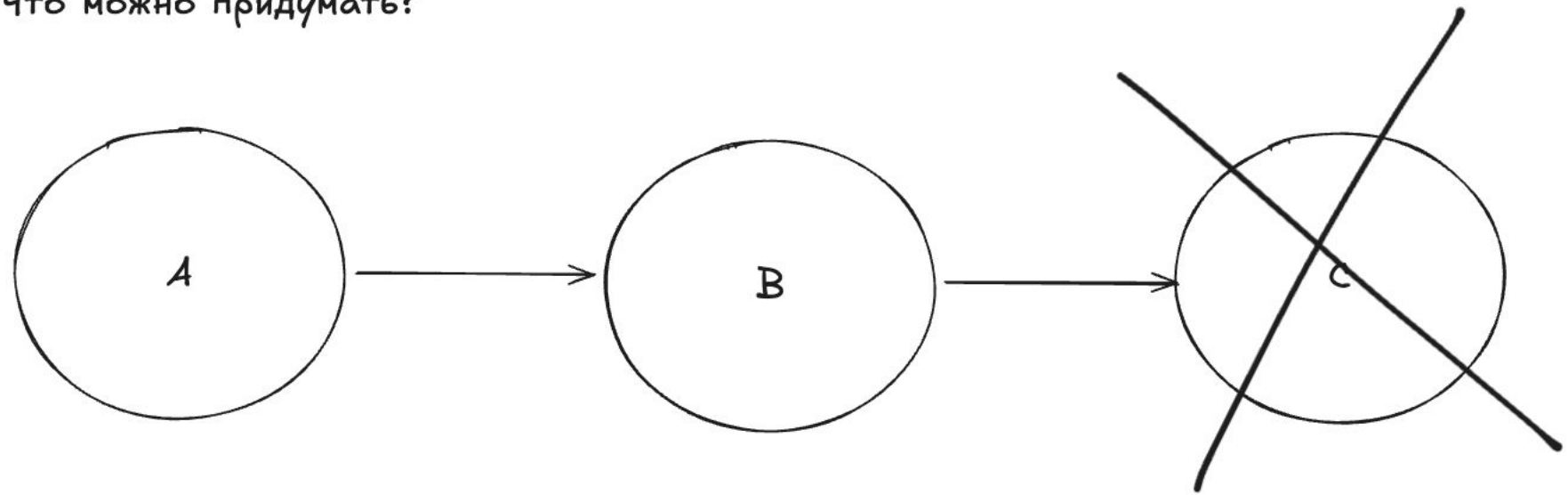
Предотвращаем отказ SPOF

Middleware.
Что можно придумать?



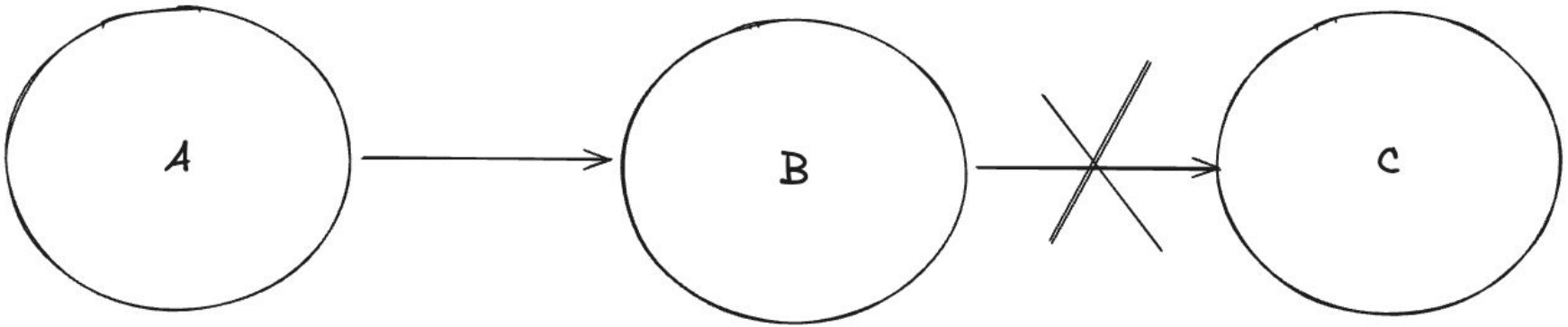
Кейс 1: связка

Кейс.
Что можно придумать?



Кейс 2: связка

Кейс.
Что можно придумать?



Микросервисы: принципы

Table 1 Relationship among microservices lifecycle stages, principles, features, and tools

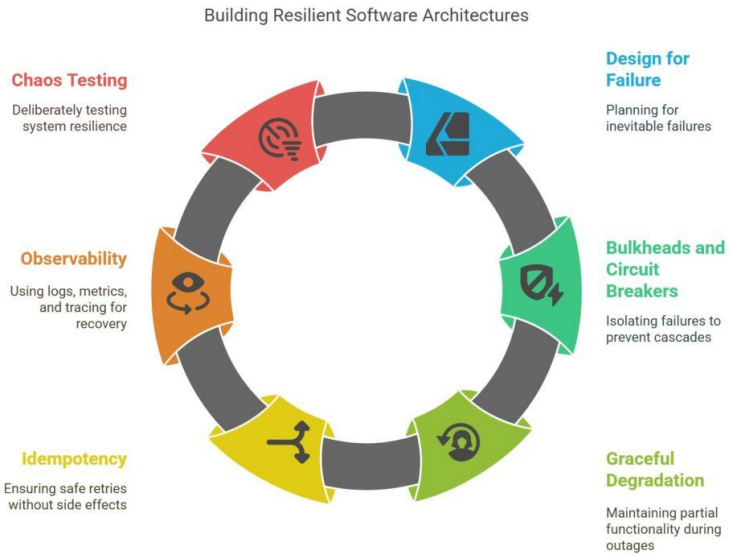
Stage	Principle	Example features	Tools/practices
Design	Modeled around business domain	Contract, business, domain, functional, interfaces, bounded context, domain-driven design, single responsibility	Domain-driven design (DDD), bounded context
Design	Hide implementation details	Bounded contexts, REST, RESTful, hide databases, data pumps, event data pumps, technology-agnostic	OpenAPI, Swagger, Kafka, RabbitMQ, Spring Cloud Data Flow
Dev	Culture of automation	Automated, automatic, continuous*(deployment, integration, delivery), environment definitions, custom images, immutable servers	Travis-CI, Chef, Ansible, CI/CD
Dev	Decentralize all	DevOps, Governance, self-service, choreography, smart endpoints, dumb pipes, database-per-service, service discovery	Zookeper, Netflix Conductor
Dev/ Ops	Isolate failure	Design for failure, failure patterns, circuit-breaker, bulkhead, timeouts, availability, consistency, antifragility,	Hystrix, Simian Army, Chaos Monkey
Ops	Deploy independently	versioning, one-service-per-host, containers	Docker, Kubernetes, canary|A/B|blue/ green testing
Ops	Highly observable	Monitoring, logging, analytics, statistics, aggregation	ELK, Elasticsearch, Logstash, Kibana

Best Practices

При проектировании: design for failure / design for reliability - процесс, благодаря которому система выполняет свои функции в заданной среде в течение ожидаемого срока службы

При разработке: reliability patterns

При тестировании: chaos engineering https://github.com/dastergon/awesome-chaos-engineering



Паттерны

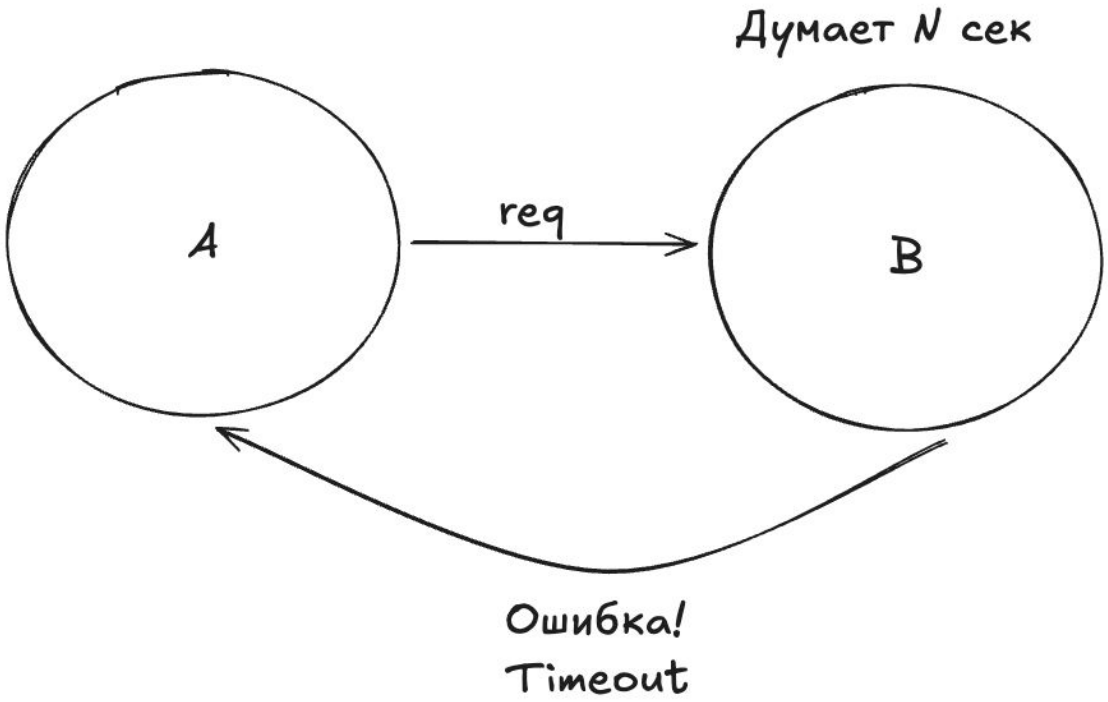
* Exponential Backoff
* Retry
* Timeout

* Circuit Breaker
* Rate Limiter
* Throttling

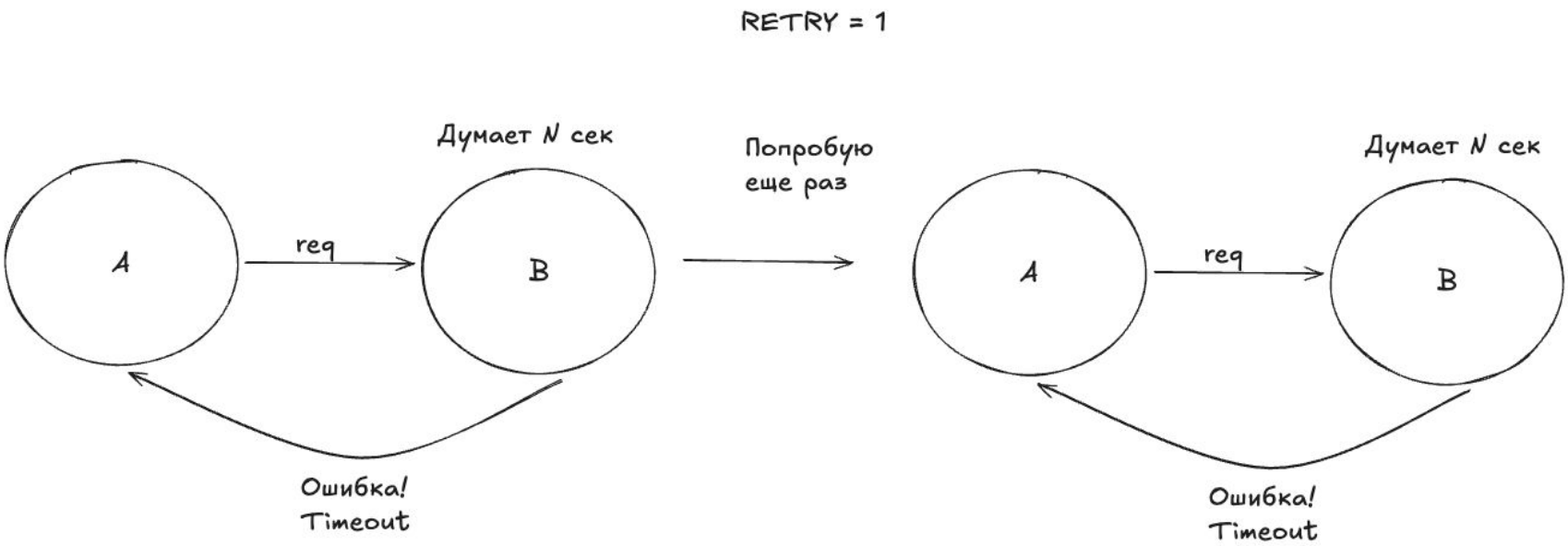
* Bulkhead
* Sidecar

* Fallback
* Graceful Degradation
* Fail fast

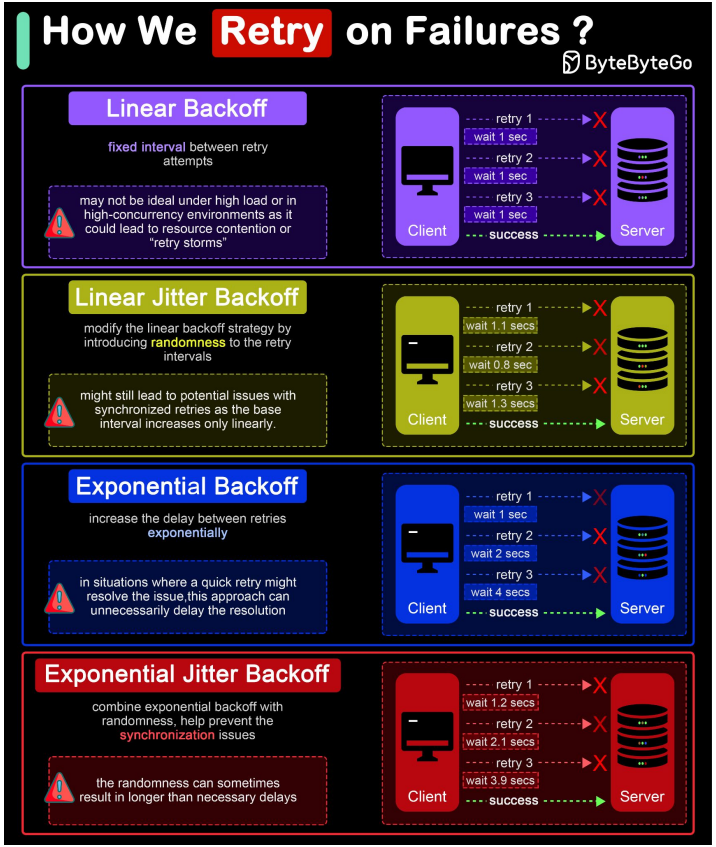
Timeout



Retry


RETRY = 1
Думает N сек
A
req
B
Ошибка!
Timeout
Попробую
еще раз
Думает N сек
A
req
B
Ошибка!
Timeout

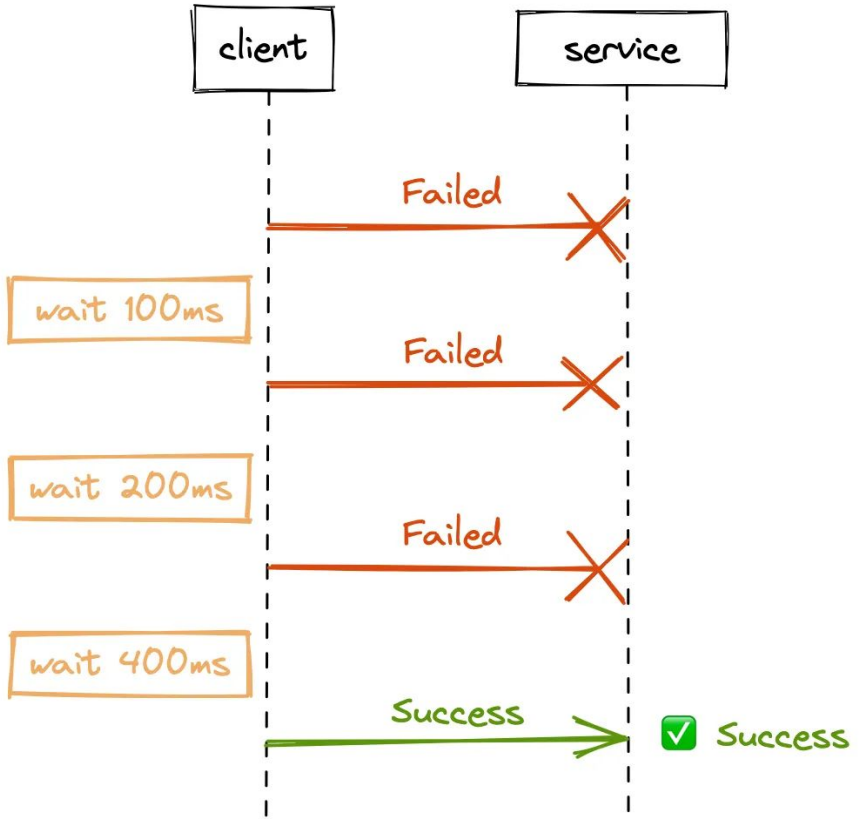
Типы Retry


How We Retry on Failures ?
ByteByteGo
Linear Backoff
fixed interval between retry attempts
may not be ideal under high load or in high-concurrency environments as it could lead to resource contention or "retry storms"
retry 1
wait 1 sec
retry 2
wait 1 sec
retry 3
wait 1 sec
success
Client
Server
Linear Jitter Backoff
modify the linear backoff strategy by introducing randomness to the retry intervals
might still lead to potential issues with synchronized retries as the base interval increases only linearly.
retry 1
wait 1.1 secs
retry 2
wait 0.8 sec
retry 3
wait 1.3 secs
success
Client
Server
Exponential Backoff
increase the delay between retries exponentially
in situations where a quick retry might resolve the issue,this approach can unnecessarily delay the resolution
retry 1
wait 1 sec
retry 2
wait 2 secs
retry 3
wait 4 secs
success
Client
Server
Exponential Jitter Backoff
combine exponential backoff with randomness, help prevent the synchronization issues
the randomness can sometimes result in longer than necessary delays
retry 1
wait 1.2 secs
retry 2
wait 2.1 secs
retry 3
wait 3.9 secs
success
Client
Server

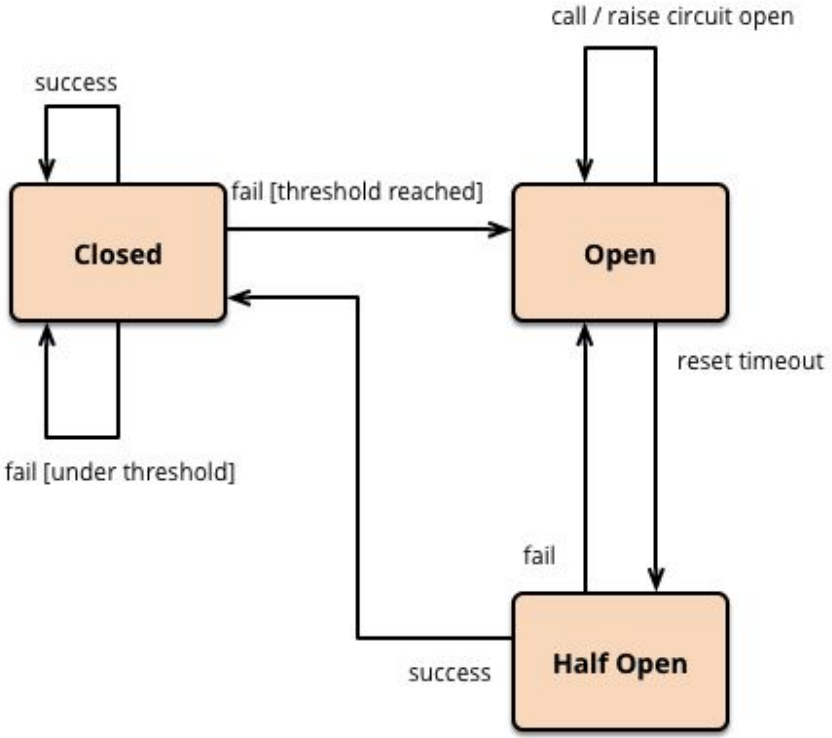
Exponential Backoff

delay = min(base_delay * (2 ** attempt), max_delay)

Для base_delay=0.5 и max_retries=5:
Попытка 1: 1.0 сек (0.5×2^1)
Попытка 2: 2.0 сек (0.5×2^2)
Попытка 3: 4.0 сек (0.5×2^3)
Попытка 4: 8.0 сек (0.5×2^4)
Попытка 5: 10.0 сек (ограничено max_delay=10)



Circuit Breaker



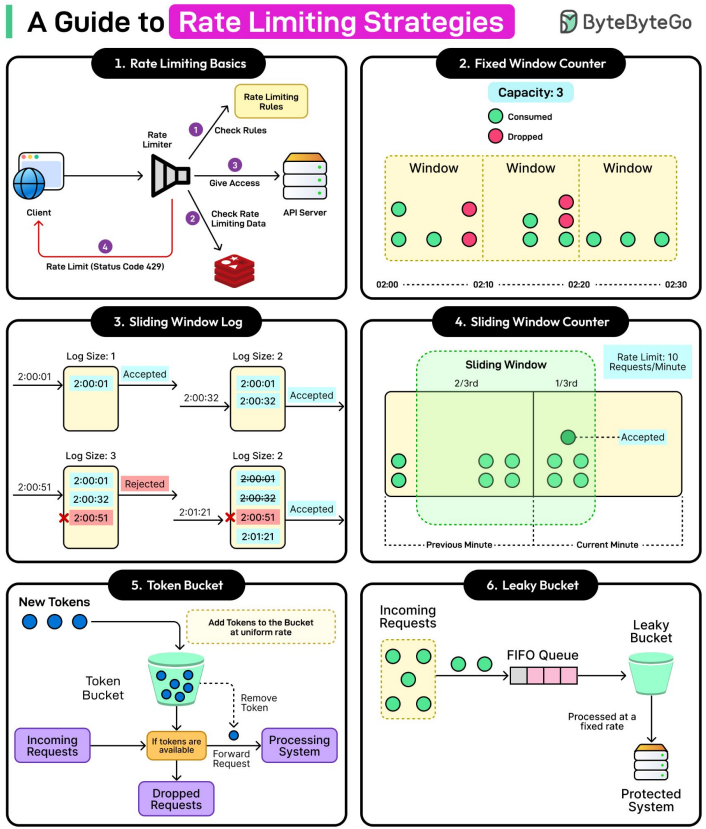
Паттерны

* Exponential Backoff
* Retry
* Timeout

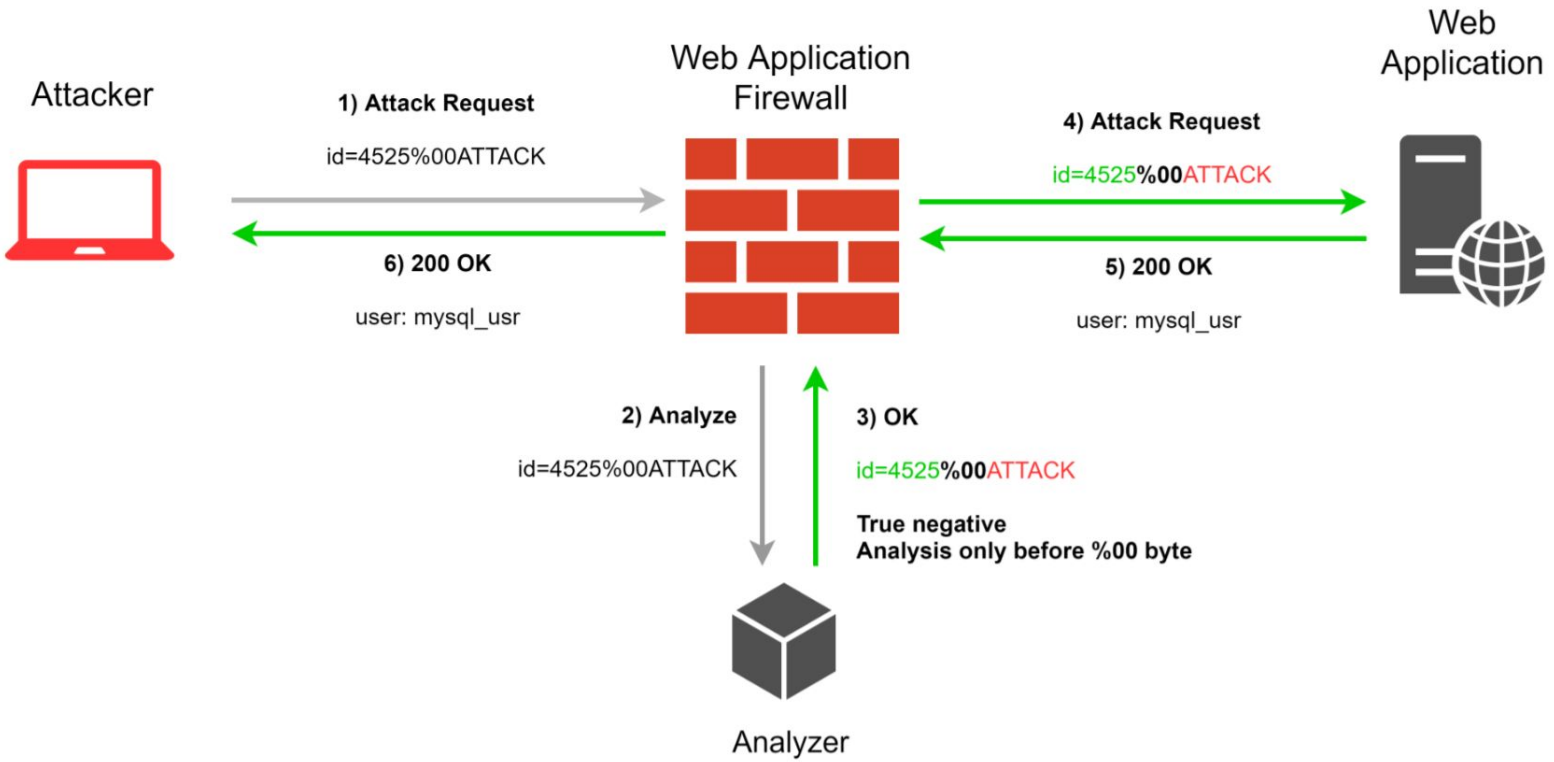
* Circuit Breaker
* Rate Limiter
* Throttling

* Fallback
* Graceful Degradation
* Fail fast

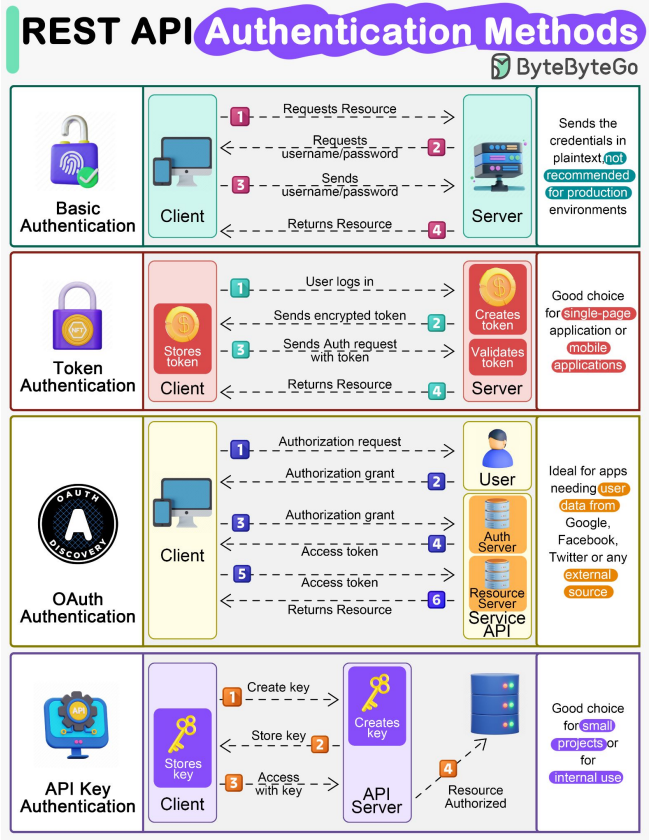
Rate Limiter



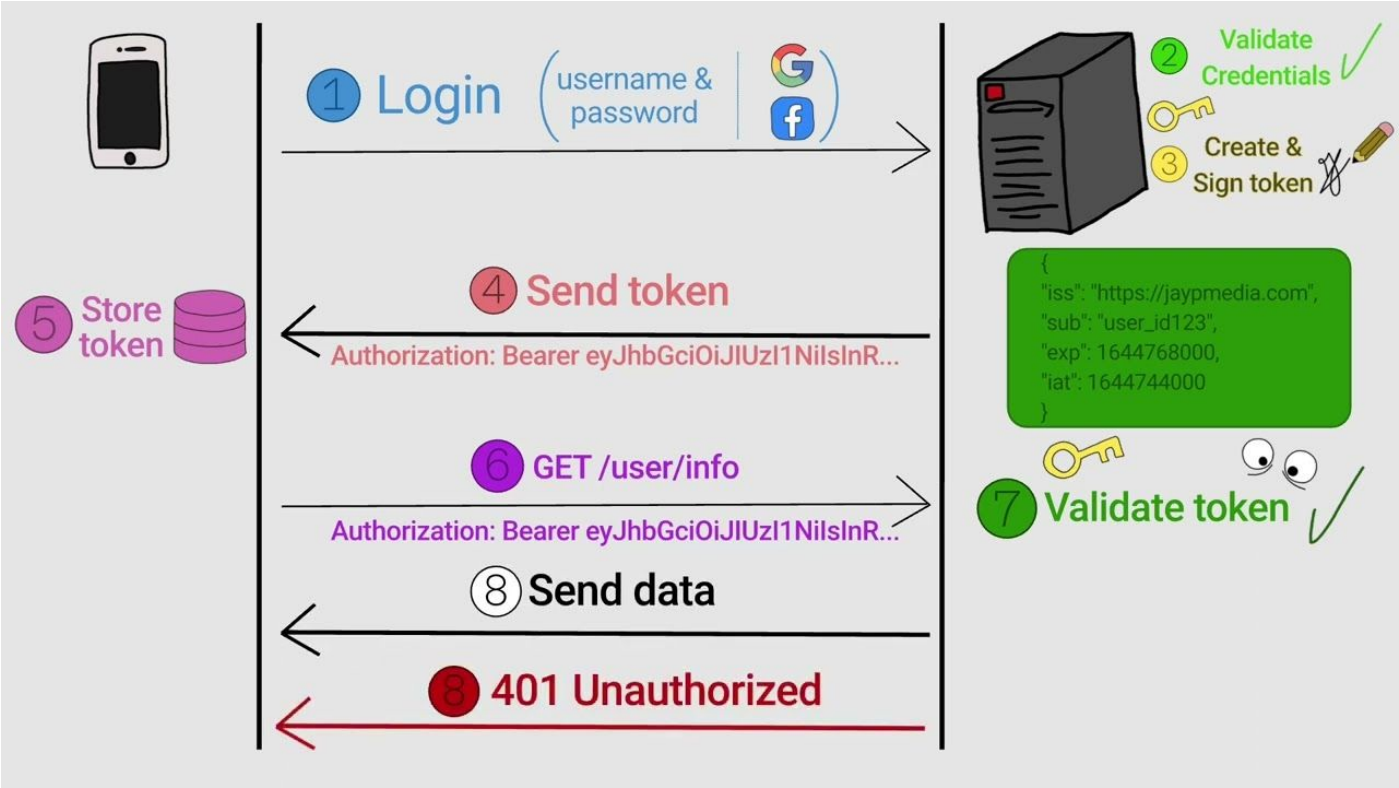
WAF


Attacker
1) Attack Request
id=4525%00ATTACK
6) 200 OK
user: mysql_usr
Web Application Firewall
4) Attack Request
id=4525%00ATTACK
5) 200 OK
user: mysql_usr
Web Application
2) Analyze
id=4525%00ATTACK
3) OK
id=4525%00ATTACK
True negative
Analysis only before %00 byte
Analyzer

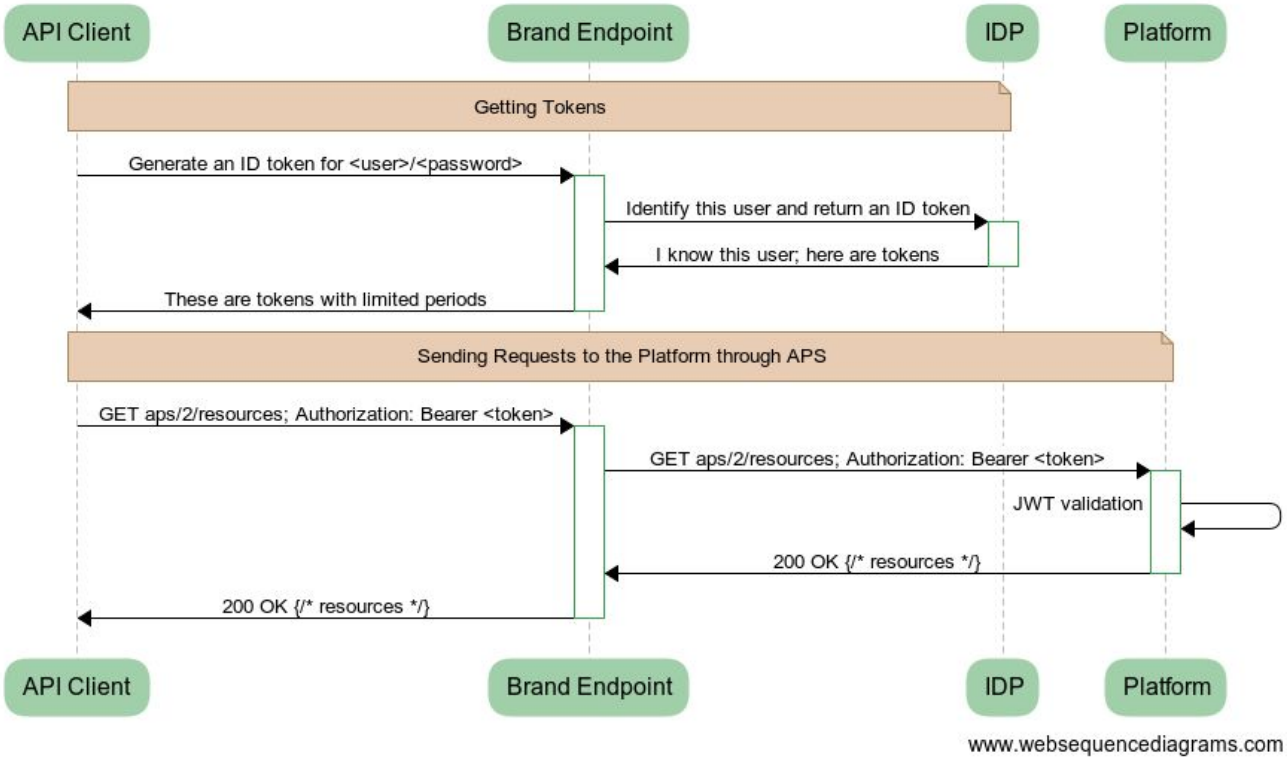
API Auth


REST API Authentication Methods
ByteByteGo
Basic Authentication
Client
1 Requests Resource
2 Requests username/password
3 Sends username/password
4 Returns Resource
Server
Sends the credentials in plaintext,not recommended for production environments
Token Authentication
Stores token
Client
1 User logs in
2 Sends encrypted token
3 Sends Auth request with token
4 Returns Resource
Creates token
Validates token
Server
Good choice for single-page application or mobile applications
OAuth Authentication
Client
1 Authorization request
2 Authorization grant
3 Authorization grant
4 Access token
5 Access token
6 Returns Resource
User
Auth Server
Resource Server
Service API
Ideal for apps needing user data from Google, Facebook, Twitter or any external source
API Key Authentication
Stores key
Client
1 Create key
2 Store key
3 Access with key
Creates key
API Server
4 Resource Authorized
Good choice for small projects or for internal use

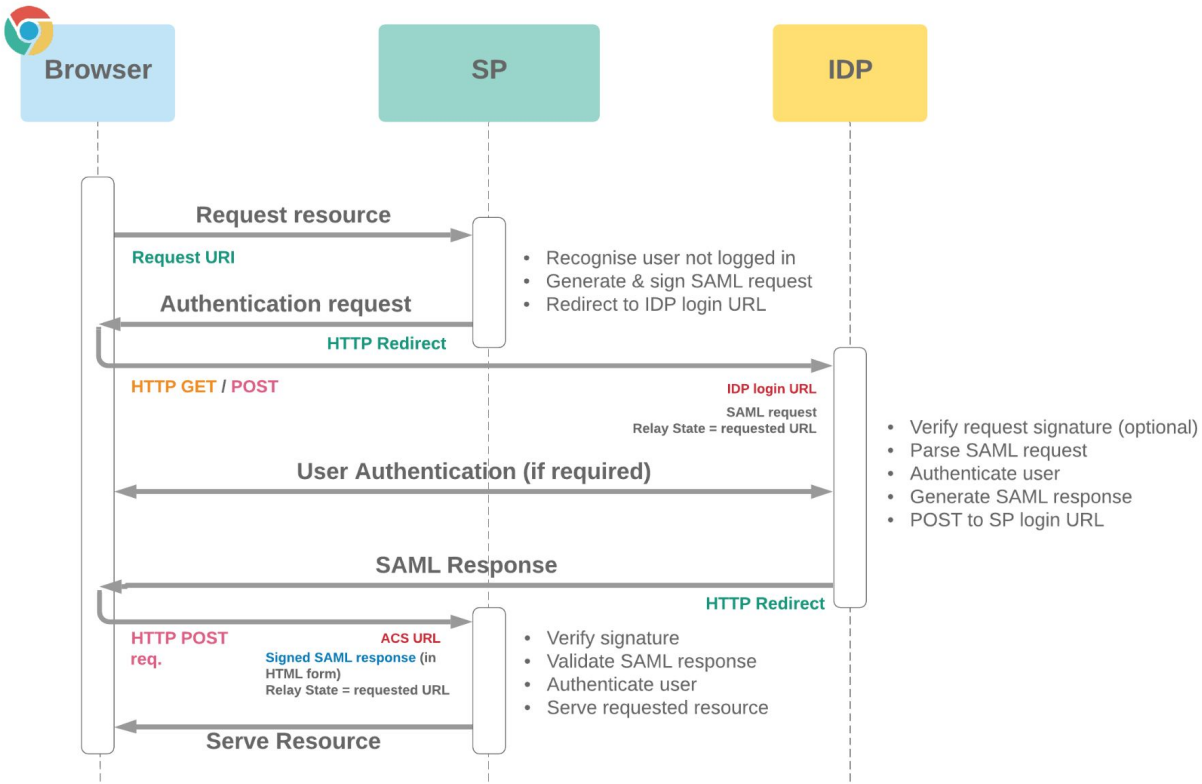
JWT


1 Login
username & password
2 Validate Credentials
3 Create & Sign token
{
"iss": "https://jaypmedia.com",
"sub": "user_id123",
"exp": 1644768000,
"iat": 1644744000
}
4 Send token
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...
5 Store token
6 GET /user/info
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...
7 Validate token
8 Send data
8 401 Unauthorized

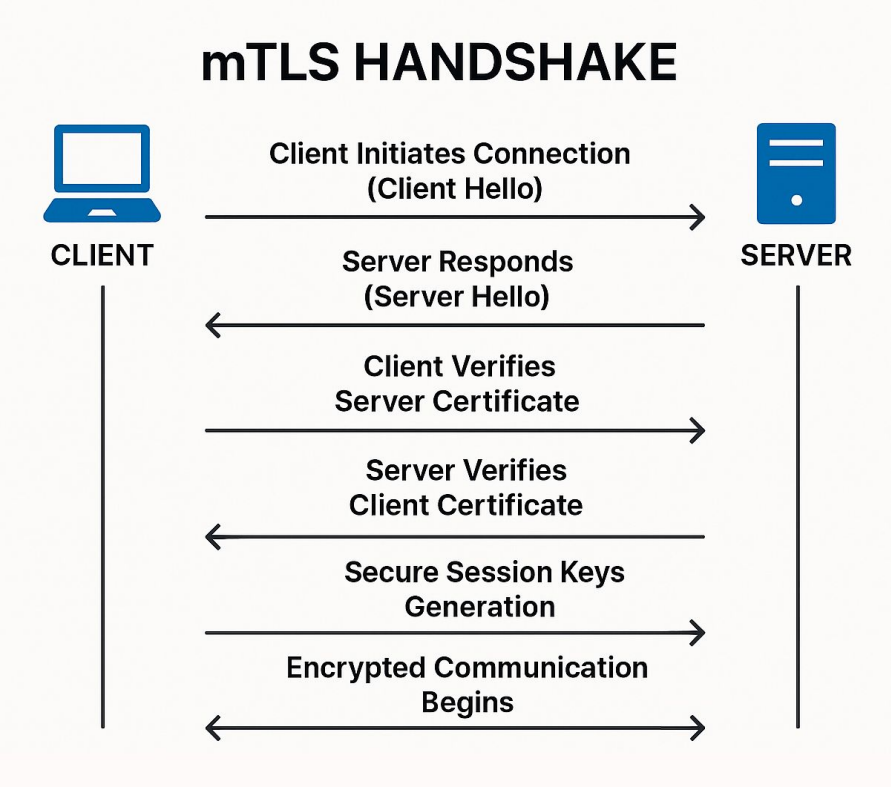
IdP


API Client
Brand Endpoint
IDP
Platform
Getting Tokens
Generate an ID token for <user>/<password>
Identify this user and return an ID token
I know this user; here are tokens
These are tokens with limited periods
Sending Requests to the Platform through APS
GET aps/2/resources; Authorization: Bearer <token>
GET aps/2/resources; Authorization: Bearer <token>
JWT validation
200 OK {/* resources */}
200 OK {/* resources */}
API Client
Brand Endpoint
IDP
Platform
www.websequencediagrams.com

IdP (SSO)


Browser
SP
IDP
Request resource
Request URI
Recognise user not logged in
Generate & sign SAML request
Redirect to IDP login URL
Authentication request
HTTP Redirect
HTTP GET / POST
IDP login URL
SAML request
Relay State = requested URL
Verify request signature (optional)
Parse SAML request
Authenticate user
Generate SAML response
POST to SP login URL
User Authentication (if required)
SAML Response
HTTP Redirect
HTTP POST req.
ACS URL
Signed SAML response (in HTML form)
Relay State = requested URL
Verify signature
Validate SAML response
Authenticate user
Serve requested resource
Serve Resource

mTLS


mTLS HANDSHAKE
CLIENT
SERVER
Client Initiates Connection
(Client Hello)
Server Responds
(Server Hello)
Client Verifies
Server Certificate
Server Verifies
Client Certificate
Secure Session Keys
Generation
Encrypted Communication
Begins

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ