Deep Dive: SPIFFE and SPIRE

By Emiliano Berenbaum and Andrew Harding

emiliano@scytale.io
harding@scytale.io

Agenda

- Federation
- JWT Support
- Envoy Demo
- Next Steps

Scytale’s Jenny Schaffer would love to talk with you!
about:

- SPIFFE/SPIRE usability issues
- SPIFFE/SPIRE documentation / information architecture / personas and roles
- Kubernetes-specific information about the above

Please email her: **jenny@scytale.io**

Introduction to SPIFFE
Andrew Jessup and
Dan Feldman

https://sched.co/HtJu

Scrutinizing SPIRE to Sensible Strengthen SPIFFE Security

Matt Moyer and Evan Gilman

https://sched.co/GrZZ

Federation

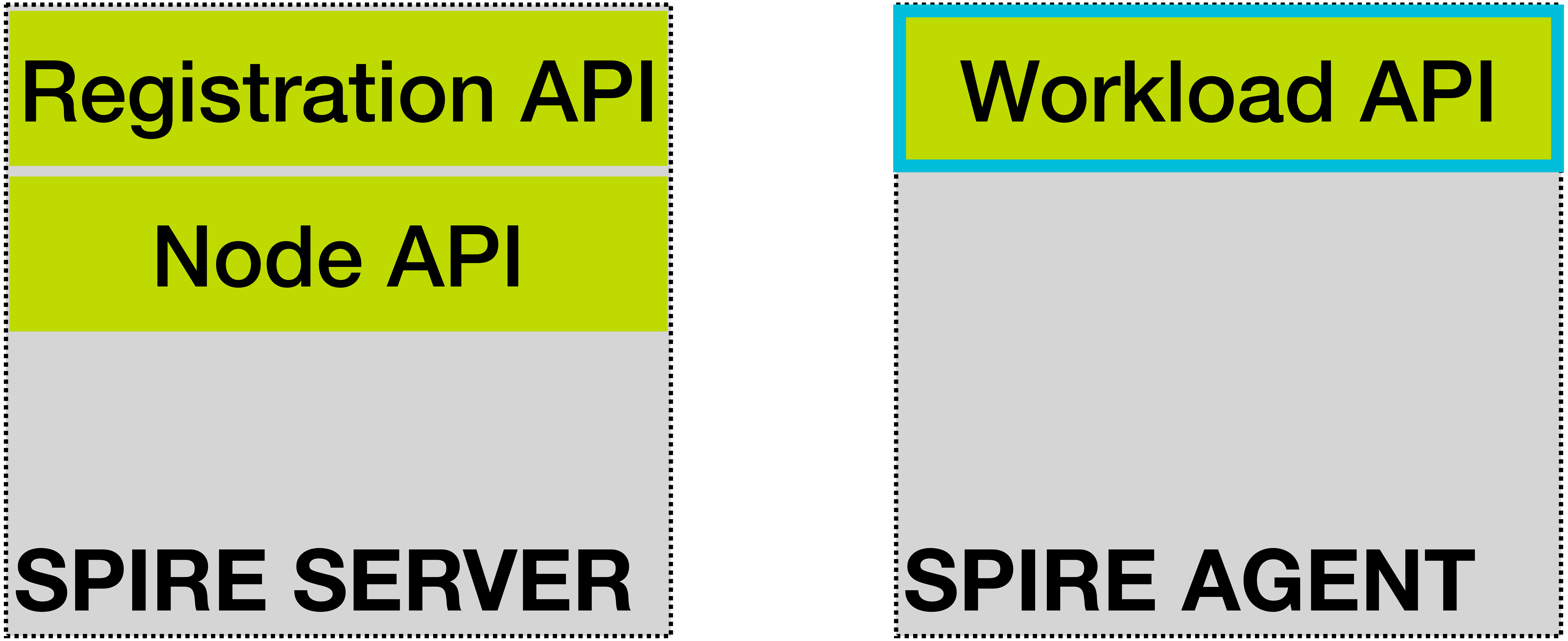
Dan Feldmans Blog Post on Federation

https://blog.scytale.io

We can set up Federation using the Registration API today

Trust bundles are exposed via the Workload API

Current Support

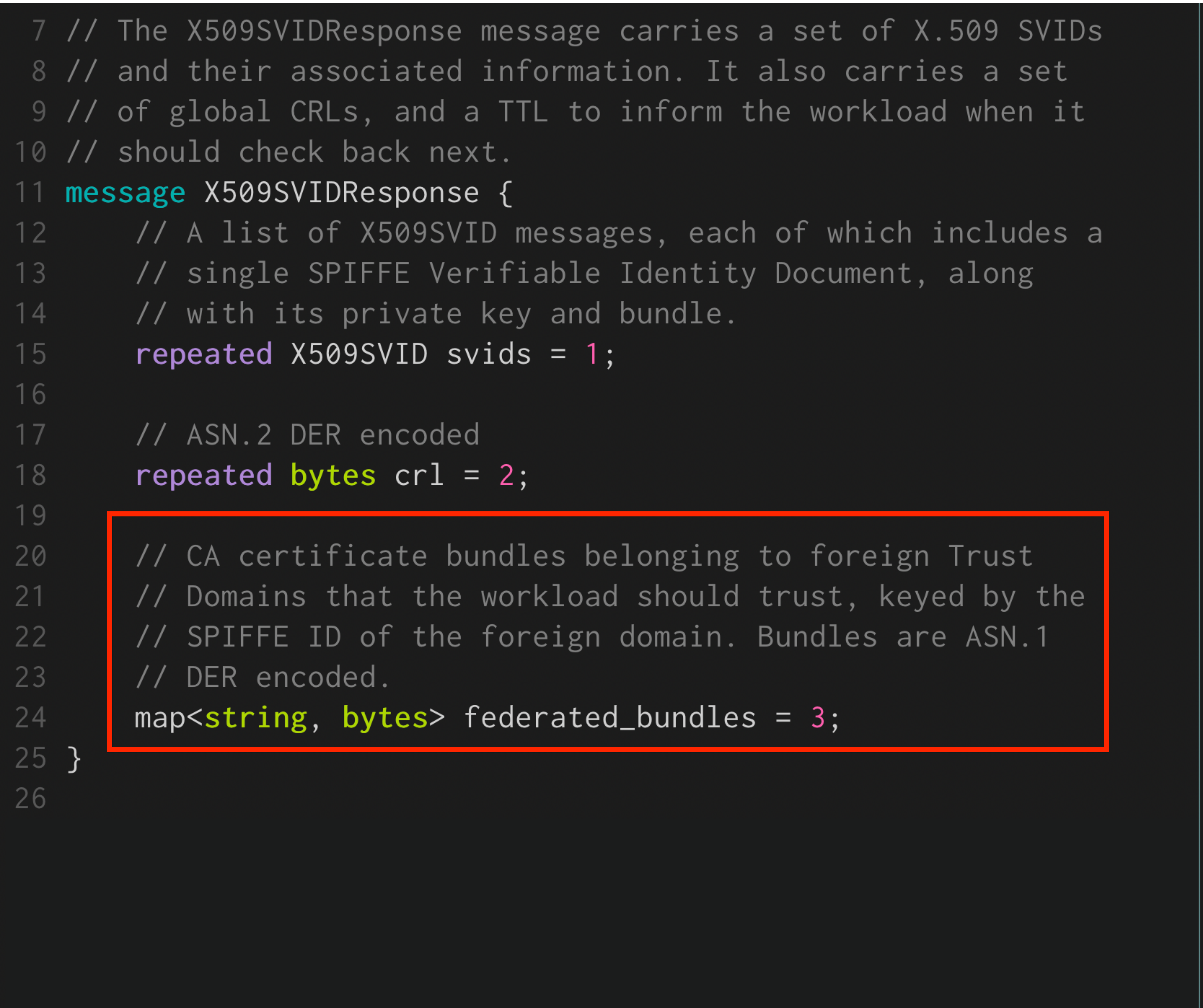


Federation Extensions

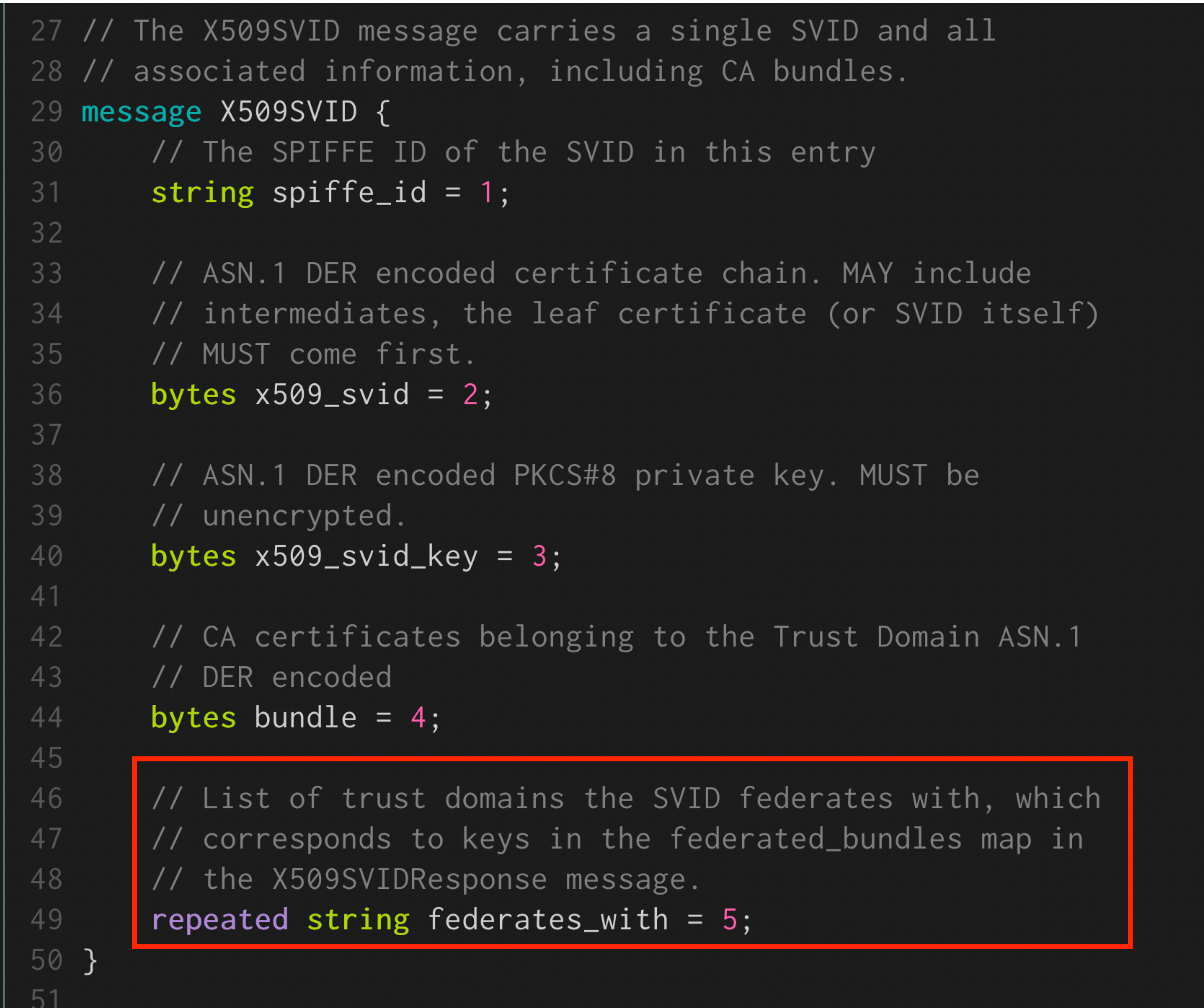
```
// The X509SVIDResponse message carries a set of X.509 SVIDs
// and their associated information. It also carries a set
// of global CRLs, and a TTL to inform the workload when it
// should check back next.
message X509SVIDResponse {
    // A list of X509SVID messages, each of which includes a
    // single SPIFFE Verifiable Identity Document, along
    // with its private key and bundle.
    repeated X509SVID svids = 1;

    // ASN.2 DER encoded
    repeated bytes crl = 2;

    // CA certificate bundles belonging to foreign Trust
    // Domains that the workload should trust, keyed by the
    // SPIFFE ID of the foreign domain. Bundles are ASN.1
    // DER encoded.
    map<string, bytes> federated_bundles = 3;
}

```

```
// The X509SVID message carries a single SVID and all
// associated information, including CA bundles.
message X509SVID {
    // The SPIFFE ID of the SVID in this entry
    string spiffe_id = 1;

    // ASN.1 DER encoded certificate chain. MAY include
    // intermediates, the leaf certificate (or SVID itself)
    // MUST come first.
    bytes x509_svid = 2;

    // ASN.1 DER encoded PKCS#8 private key. MUST be
    // unencrypted.
    bytes x509_svid_key = 3;

    // CA certificates belonging to the Trust Domain ASN.1
    // DER encoded
    bytes bundle = 4;

    // List of trust domains the SVID federates with, which
    // corresponds to keys in the federated_bundles map in
    // the X509SVIDResponse message.
    repeated string federates_with = 5;
}
```

Federation Extensions

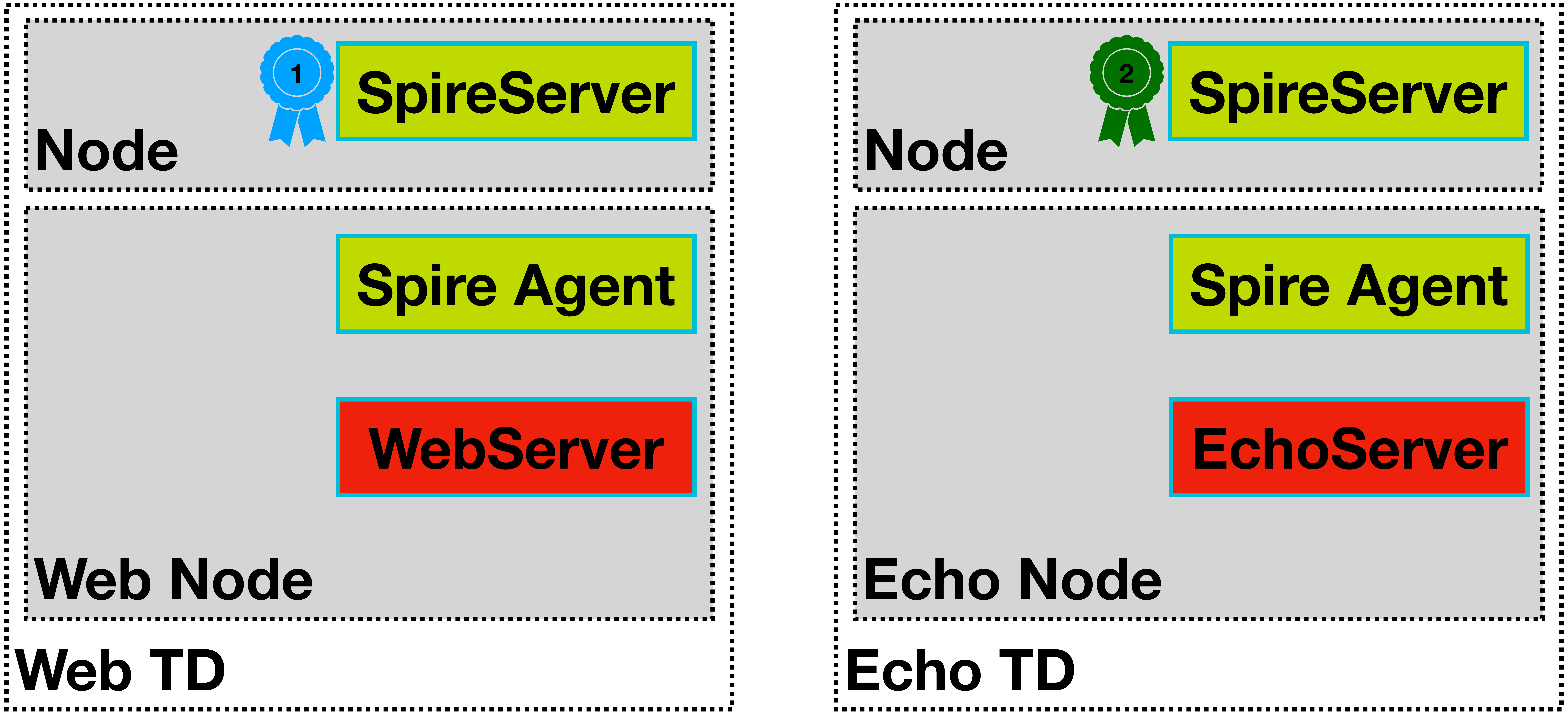


```
// The X509SVIDResponse message carries a set of X.509 SVIDs
// and their associated information. It also carries a set
// of global CRLs, and a TTL to inform the workload when it
// should check back next.
message X509SVIDResponse {
    // A list of X509SVID messages, each of which includes a
    // single SPIFFE Verifiable Identity Document, along
    // with its private key and bundle.
    repeated X509SVID svids = 1;

    // ASN.2 DER encoded
    repeated bytes crl = 2;

    // CA certificate bundles belonging to foreign Trust
    // Domains that the workload should trust, keyed by the
    // SPIFFE ID of the foreign domain. Bundles are ASN.1
    // DER encoded.
    map<string, bytes> federated_bundles = 3;
}
```



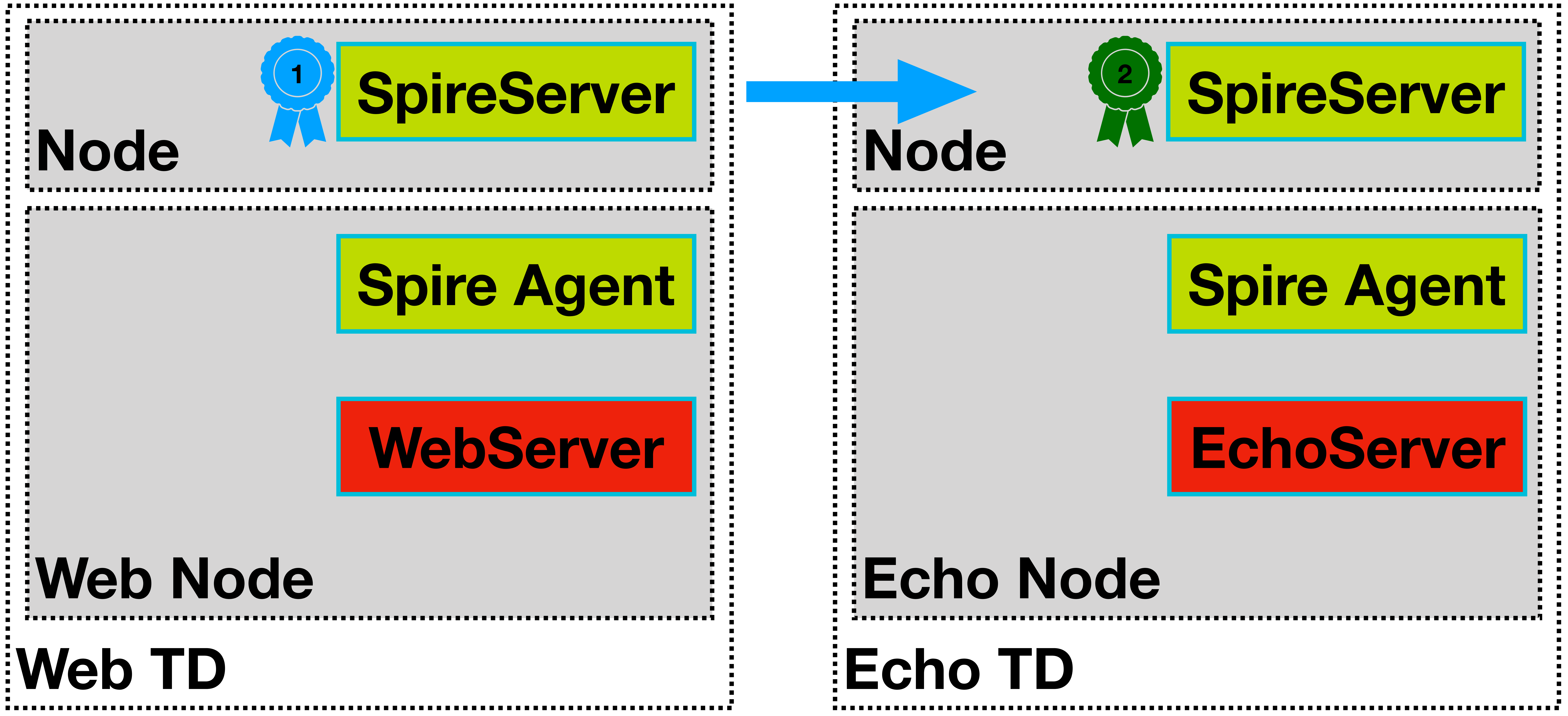
```
// The X509SVID message carries a single SVID and all
// associated information, including CA bundles.
message X509SVID {
    // The SPIFFE ID of the SVID in this entry
    string spiffe_id = 1;

    // ASN.1 DER encoded certificate chain. MAY include
    // intermediates, the leaf certificate (or SVID itself)
    // MUST come first.
    bytes x509_svid = 2;

    // ASN.1 DER encoded PKCS#8 private key. MUST be
    // unencrypted.
    bytes x509_svid_key = 3;

    // CA certificates belonging to the Trust Domain ASN.1
    // DER encoded
    bytes bundle = 4;

    // List of trust domains the SVID federates with, which
    // corresponds to keys in the federated_bundles map in
    // the X509SVIDResponse message.
    repeated string federates_with = 5;
}
```

Federation Flow

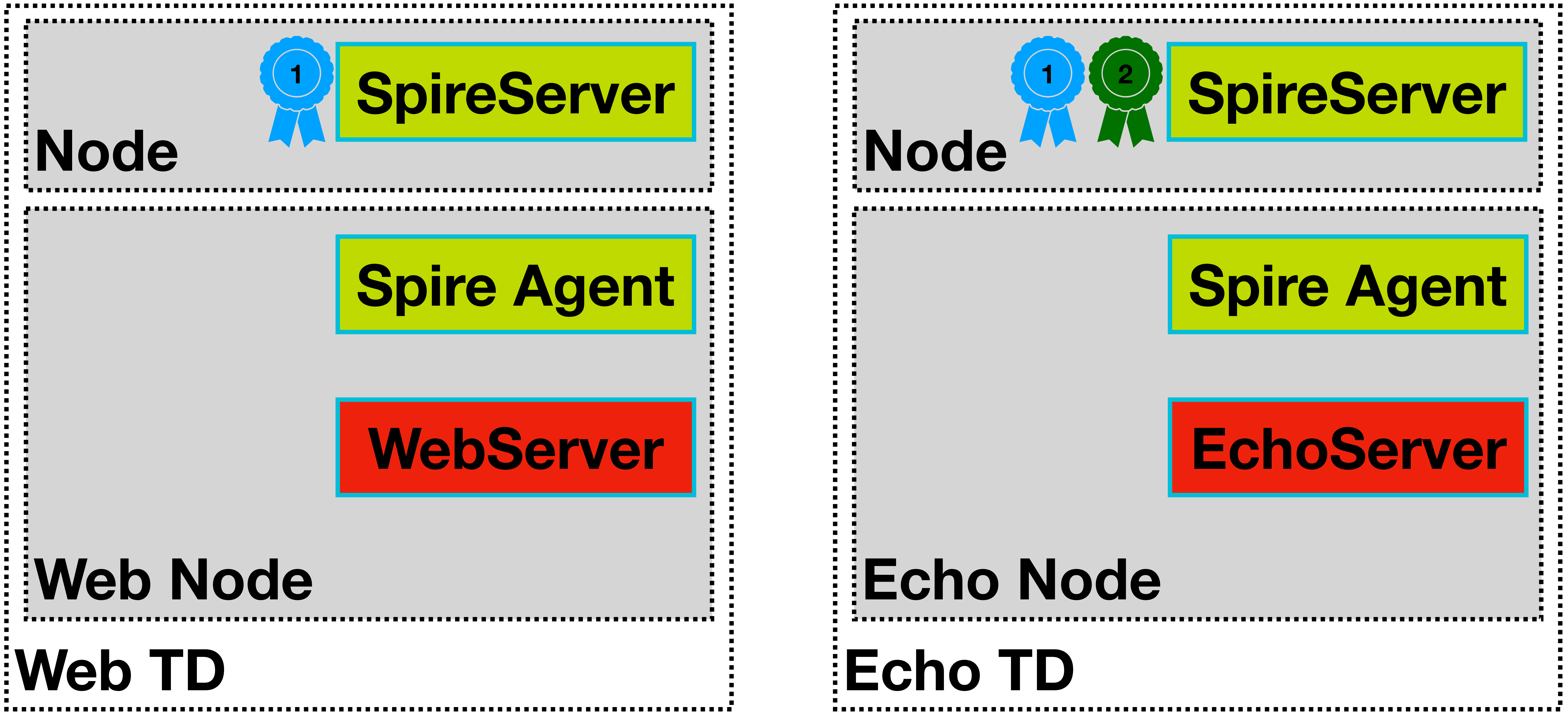


Federation Flow

Push Bundle from WebTD to EchoTD

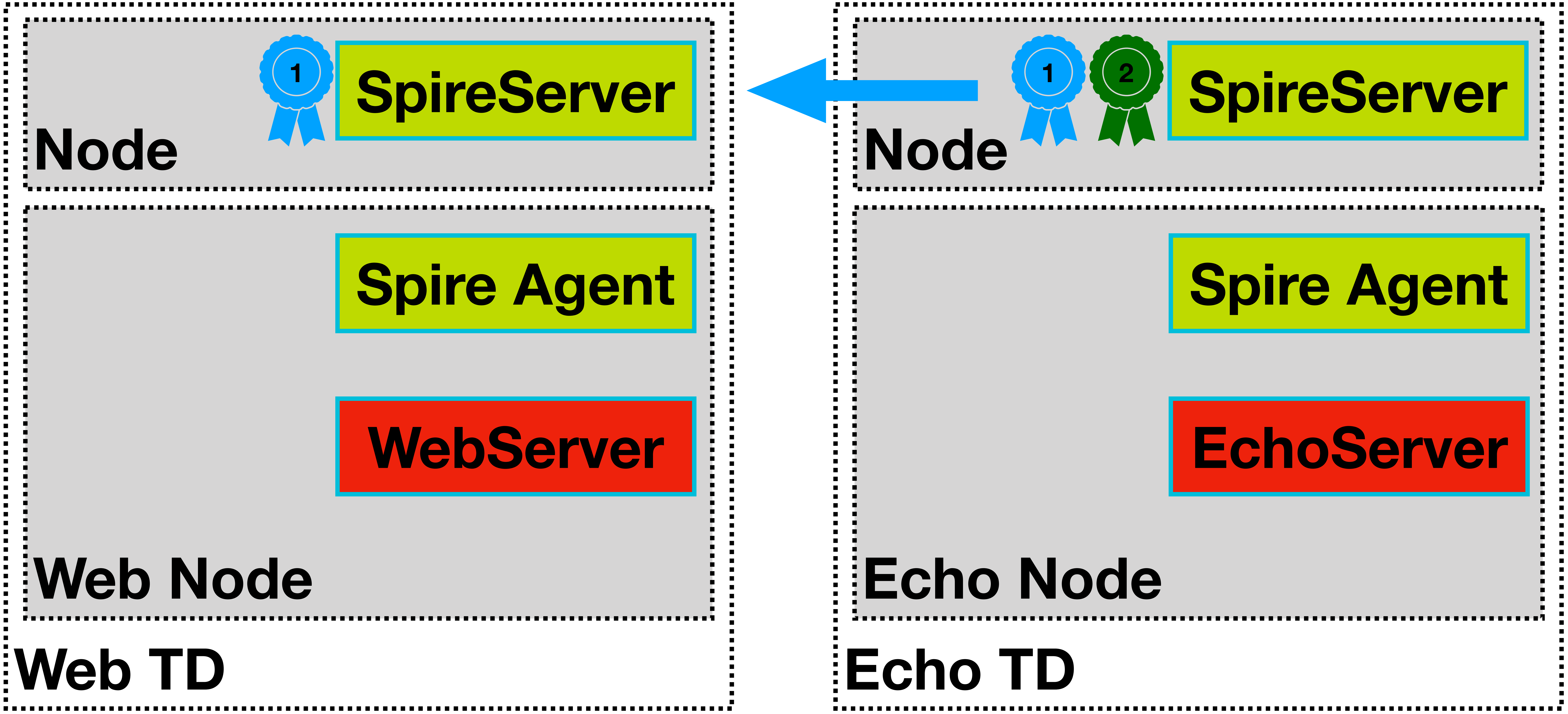


Federation Flow

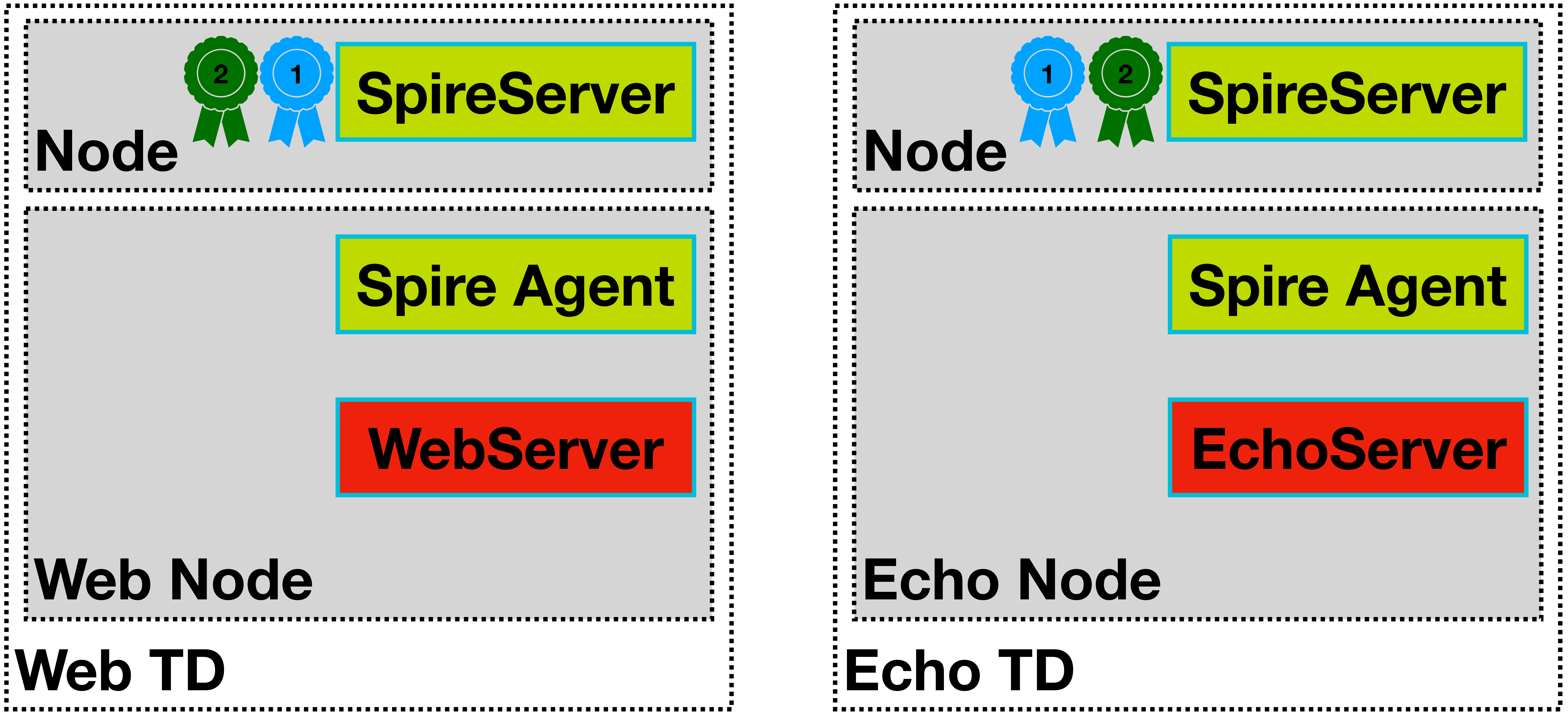


Federation Flow

Push Bundle from EchoTD to WebTD

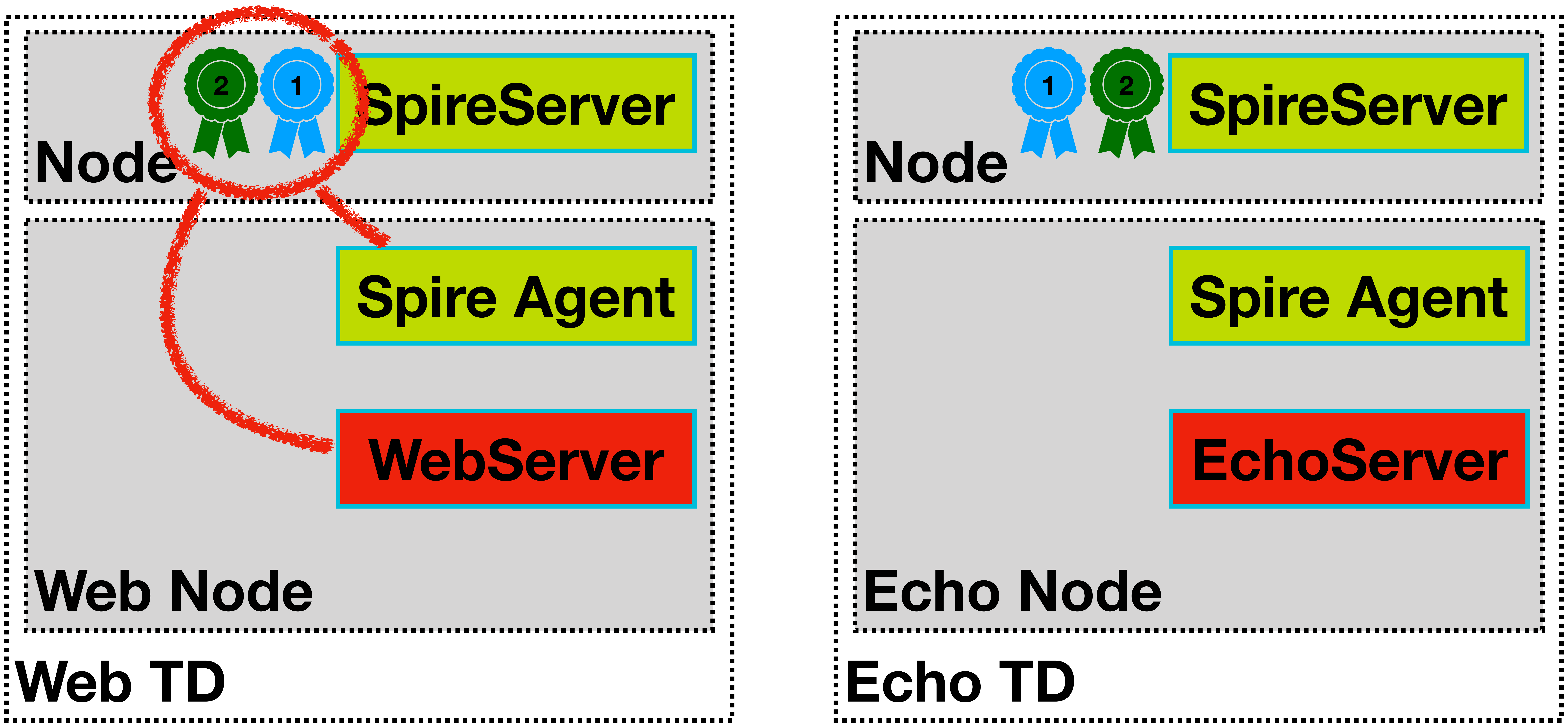


Federation Flow



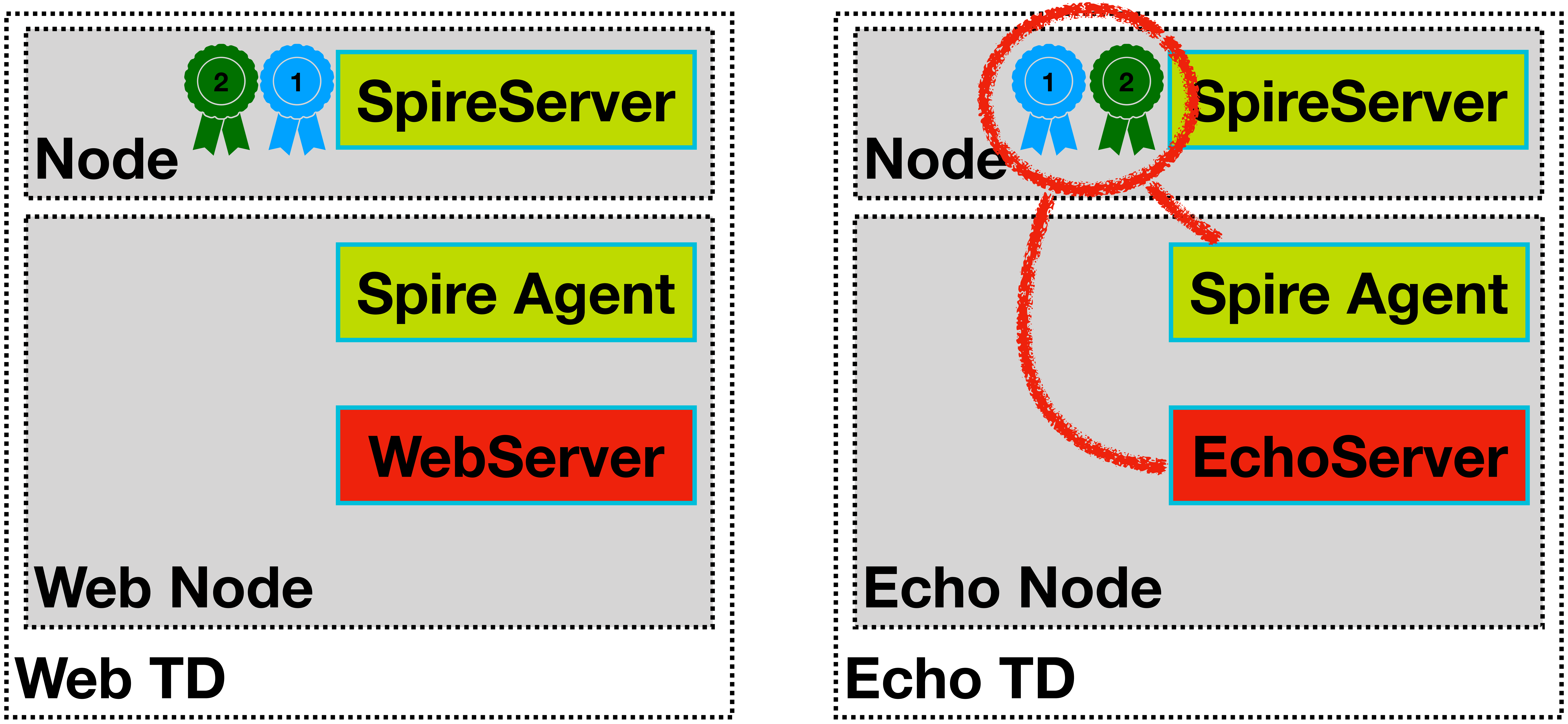
Federation Flow

Register WebServer and Federate It to the EchoTD



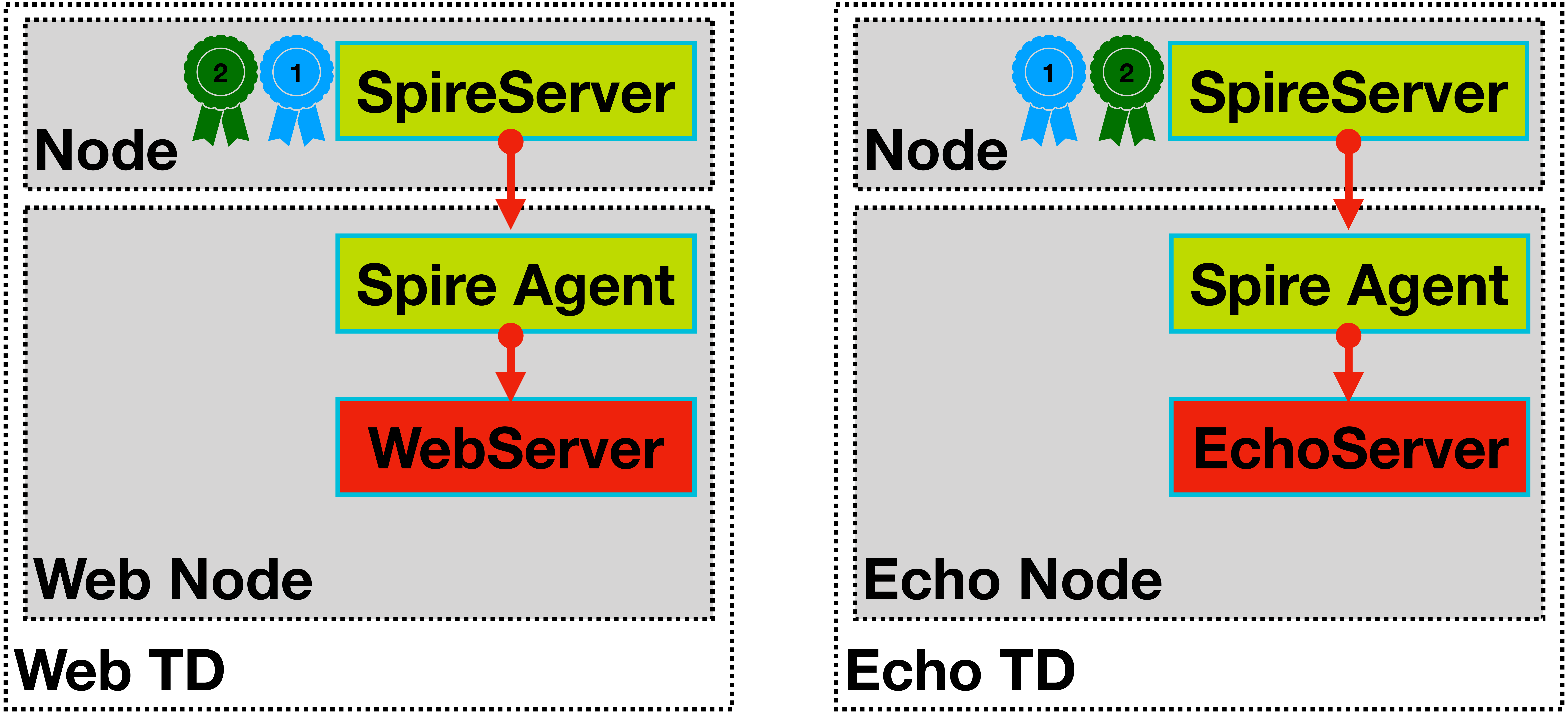
Federation Flow

Register EchoServer and Federate It to the WebTD

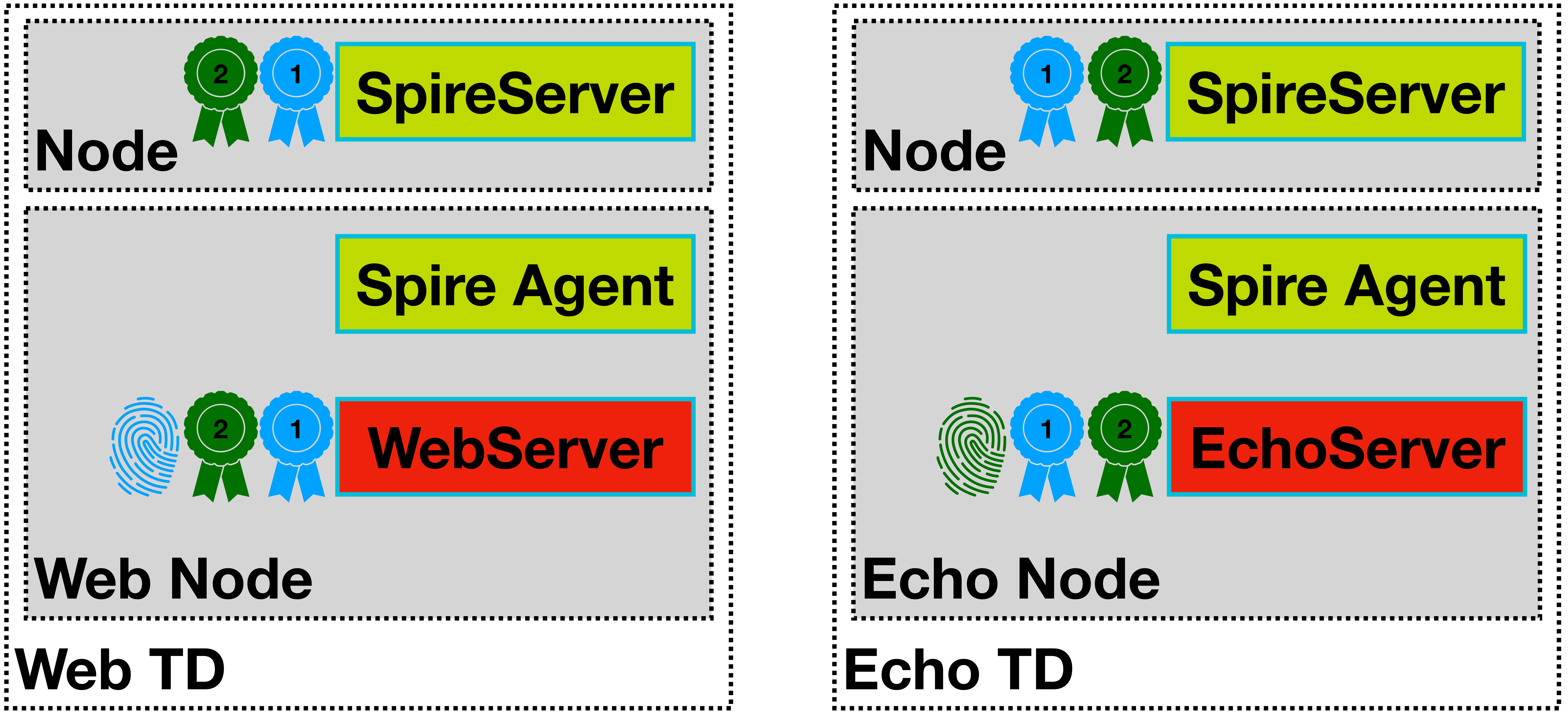


Federation Flow

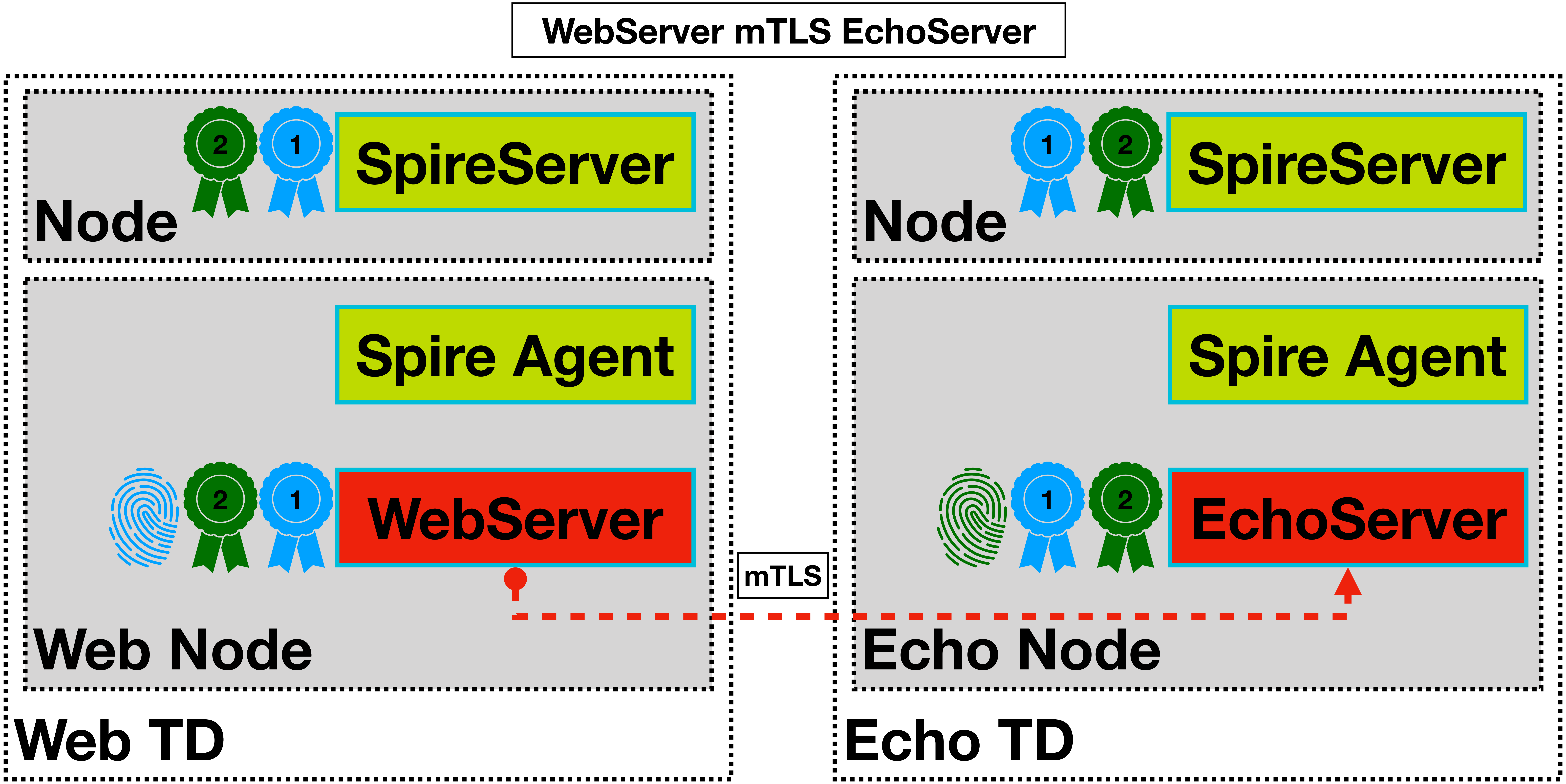
Attestation



Federation Flow

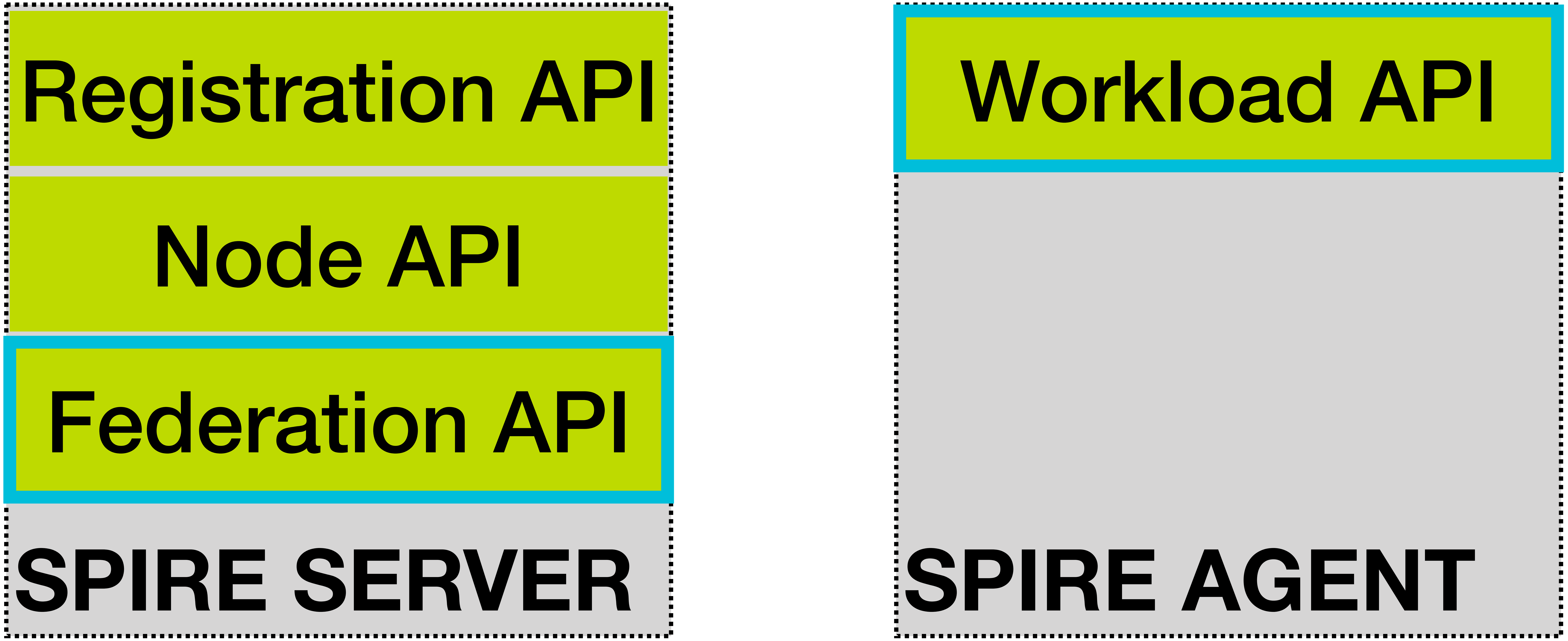


Federation Flow



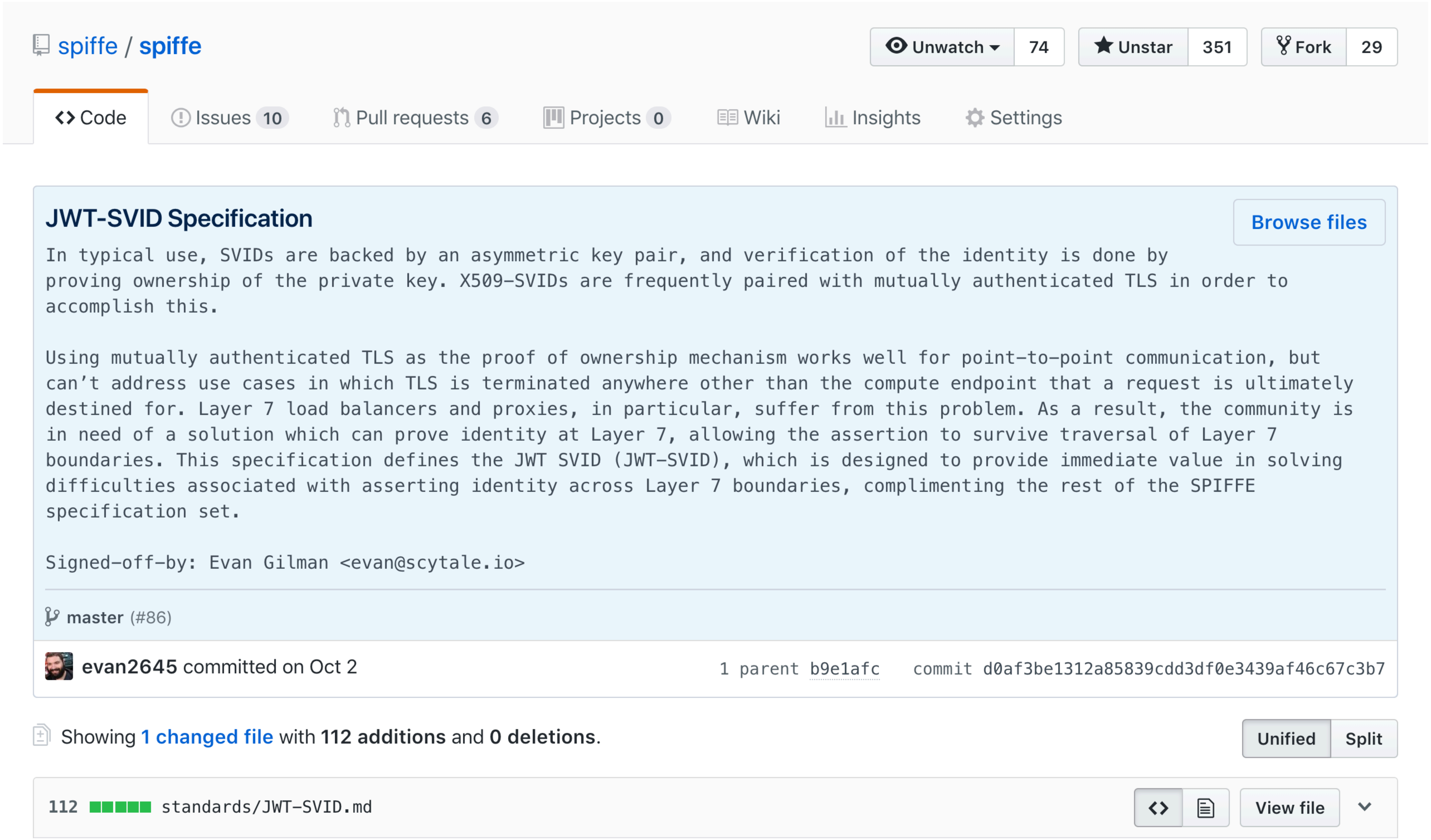
Next Step: Support for Federation API

Federation Support



Next Step:
Bundle Format

JWT



spiffe / spiffe

Unwatch 74 | Unstar 351 | Fork 29

Code | Issues 10 | Pull requests 6 | Projects 0 | Wiki | Insights | Settings

JWT-SVID Specification

Browse files

```
In typical use, SVIDs are backed by an asymmetric key pair, and verification of the identity is done by
proving ownership of the private key. X509-SVIDs are frequently paired with mutually authenticated TLS in order to
accomplish this.

Using mutually authenticated TLS as the proof of ownership mechanism works well for point-to-point communication, but
can't address use cases in which TLS is terminated anywhere other than the compute endpoint that a request is ultimately
destined for. Layer 7 load balancers and proxies, in particular, suffer from this problem. As a result, the community is
in need of a solution which can prove identity at Layer 7, allowing the assertion to survive traversal of Layer 7
boundaries. This specification defines the JWT SVID (JWT-SVID), which is designed to provide immediate value in solving
difficulties associated with asserting identity across Layer 7 boundaries, complimenting the rest of the SPIFFE
specification set.

Signed-off-by: Evan Gilman <evan@scytale.io>
```

master (#86)

evan2645 committed on Oct 2 — 1 parent b9e1afc — commit d0af3be1312a85839cdd3df0e3439af46c67c3b7

Showing 1 changed file with 112 additions and 0 deletions.

Unified | Split

112 standards/JWT-SVID.md — View file

JWT Extensions

```
message JWTSVID {
    string spiffe_id = 1;

    // Encoded using JWS Compact Serialization
    string svid = 2;
}

message JWTSVIDRequest {
    repeated string audience = 1;

    // SPIFFE ID of the JWT being requested
    // If not set, all IDs will be returned
    string spiffe_id = 2;
}

message JWTSVIDResponse {
    repeated JWTSVID svids = 1;
}

message JWTBundlesRequest { }

message JWTBundlesResponse {
    // JWK sets, keyed by trust domain URI
    map<string, bytes> bundles = 1;
}
```

```
message ValidateJWTSVIDRequest {
    string audience = 1;

    // Encoded using JWS Compact Serialization
    string svid = 3;
}

message ValidateJWTSVIDResponse {
    string spiffe_id = 1;
    google.protobuf.Struct claims = 2;
}

service SpiffeWorkloadAPI {
    // JWT-SVID Profile
    rpc FetchJWTSVID(JWTSVIDRequest)
        returns (JWTSVIDResponse);
    rpc FetchJWTBundles(JWTBundlesRequest)
        returns (stream JWTBundlesResponse);
    rpc ValidateJWTSVID(ValidateJWTSVIDRequest)
        returns (ValidateJWTSVIDResponse);

```

JWT Extensions

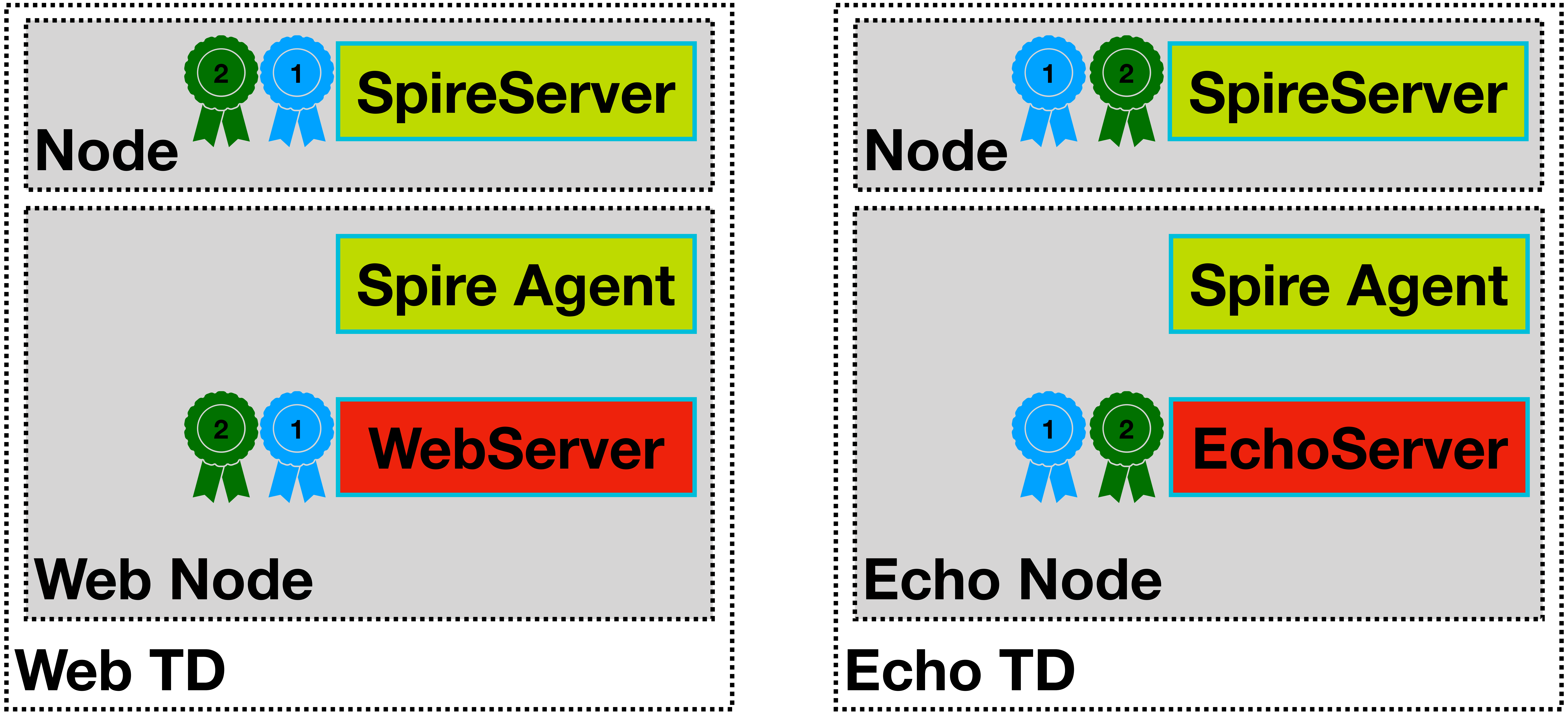
```
message JWTSVID {
    string spiffe_id = 1;

    // Encoded using JWS Compact Serialization
    string svid = 2;
}

message JWTSVIDRequest {
    repeated string audience = 1;

    // SPIFFE ID of the JWT being requested
    // If not set, all IDs will be returned
    string spiffe_id = 2;
}

message JWTSVIDResponse {
    repeated JWTSVID svids = 1;
}

message JWTBundlesRequest { }

message JWTBundlesResponse {
    // JWK sets, keyed by trust domain URI
    map<string, bytes> bundles = 1;
}
```

```
message ValidateJWTSVIDRequest {
    string audience = 1;

    // Encoded using JWS Compact Serialization
    string svid = 3;
}

message ValidateJWTSVIDResponse {
    string spiffe_id = 1;
    google.protobuf.Struct claims = 2;
}

service SpiffeWorkloadAPI {
    // JWT-SVID Profile
    rpc FetchJWTSVID(JWTSVIDRequest)
        returns (JWTSVIDResponse);
    rpc FetchJWTBundles(JWTBundlesRequest)
        returns (stream JWTBundlesResponse);
    rpc ValidateJWTSVID(ValidateJWTSVIDRequest)
        returns (ValidateJWTSVIDResponse);

```

JWT Validation

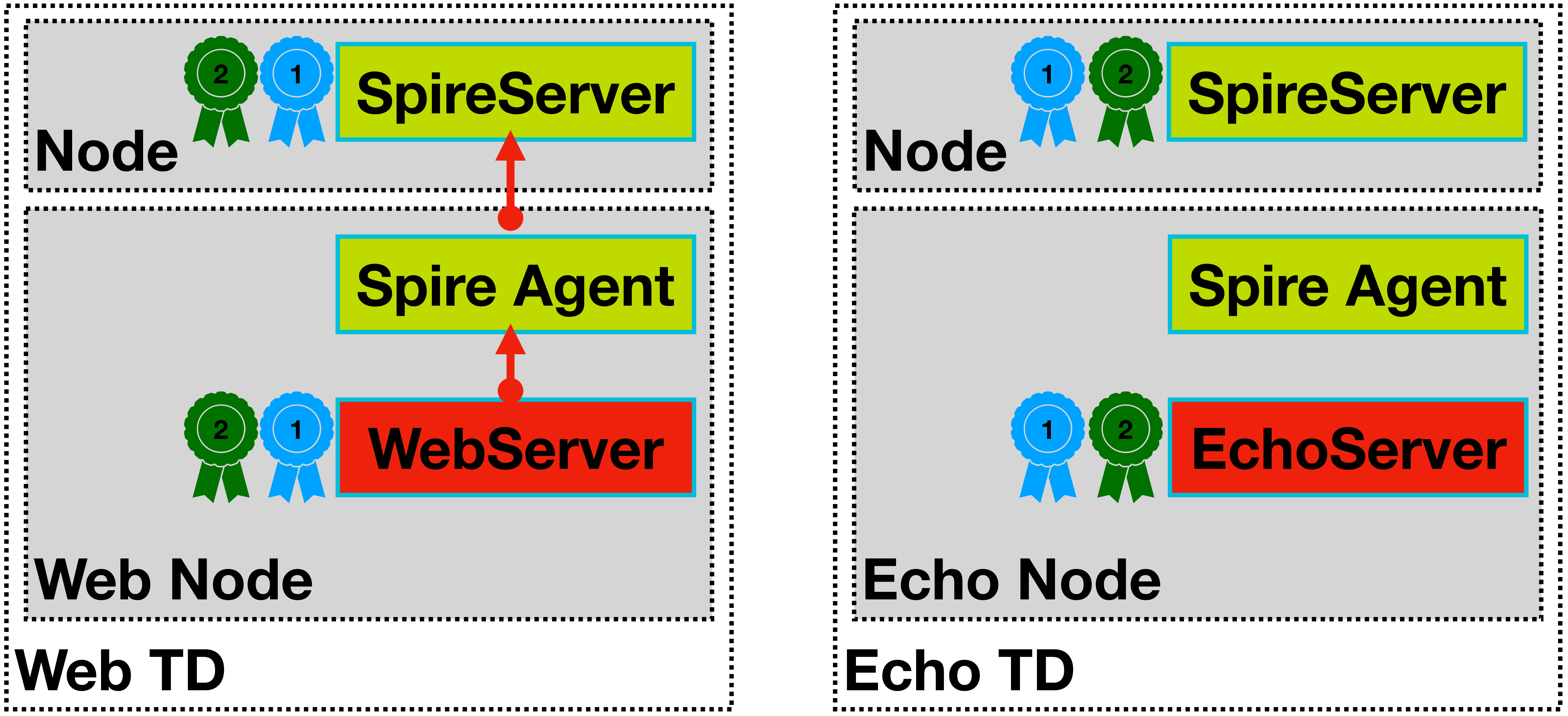
- Use the Workload API and SPIRE Agent to validate the JWT using **ValidateJWTSVID**
- Request the JWKS document (**FetchJWTBundles**) needed for validation and an external process (or existing library) will validate the JWT.

JWT Flow

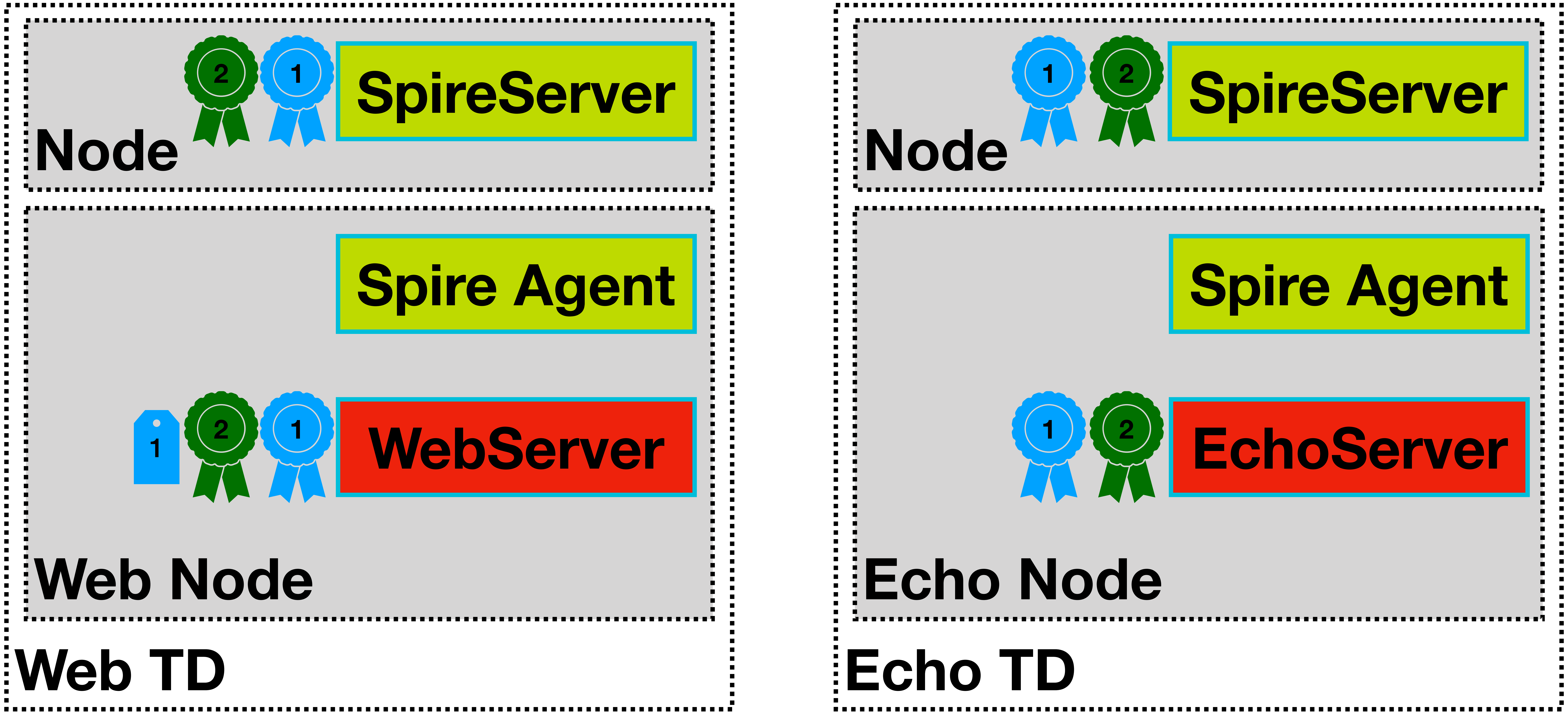

2
1
SpireServer
Node
Spire Agent
2
1
WebServer
Web Node
Web TD
1
2
SpireServer
Node
Spire Agent
1
2
EchoServer
Echo Node
Echo TD

JWT Flow

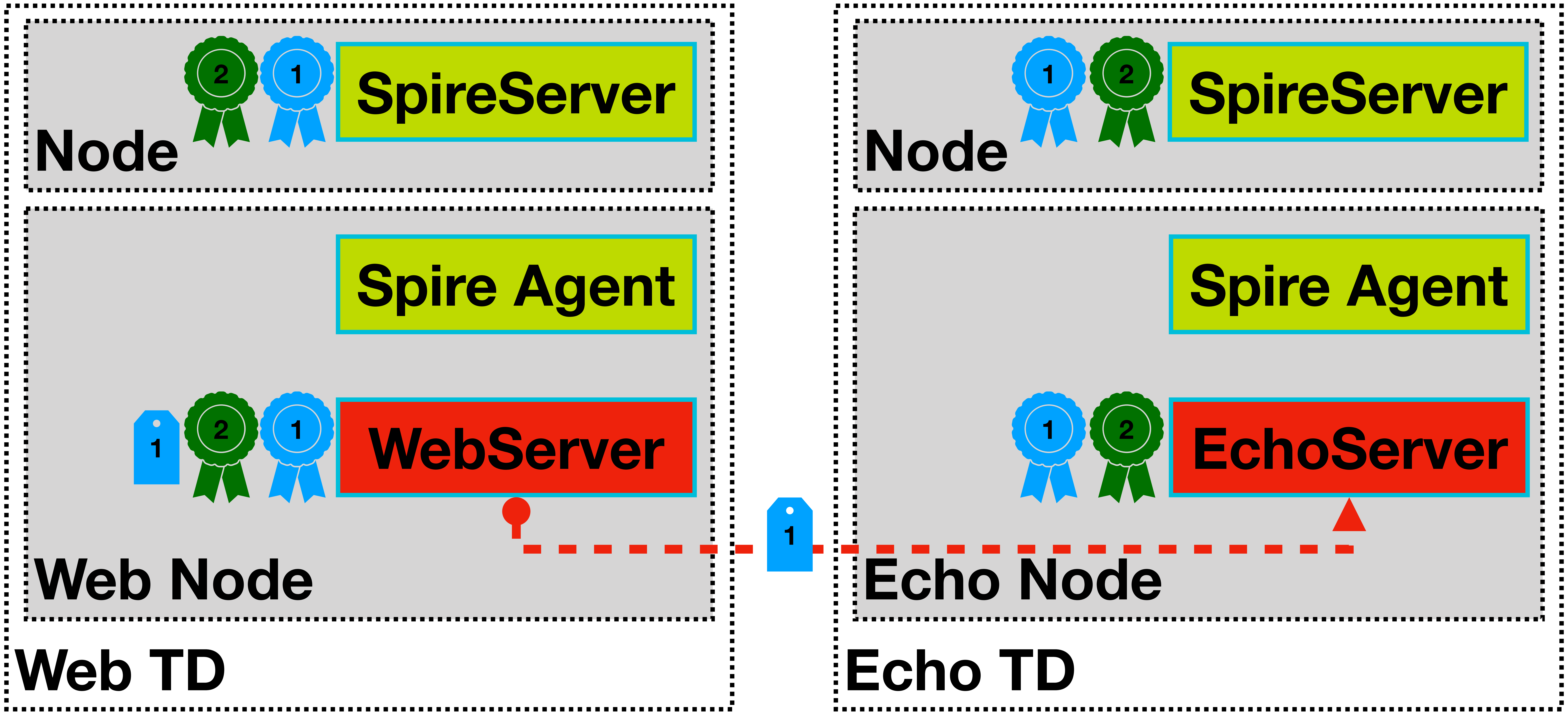
Fetch JWT (Audience = EchoServer, Subject = WebServer)



JWT Flow

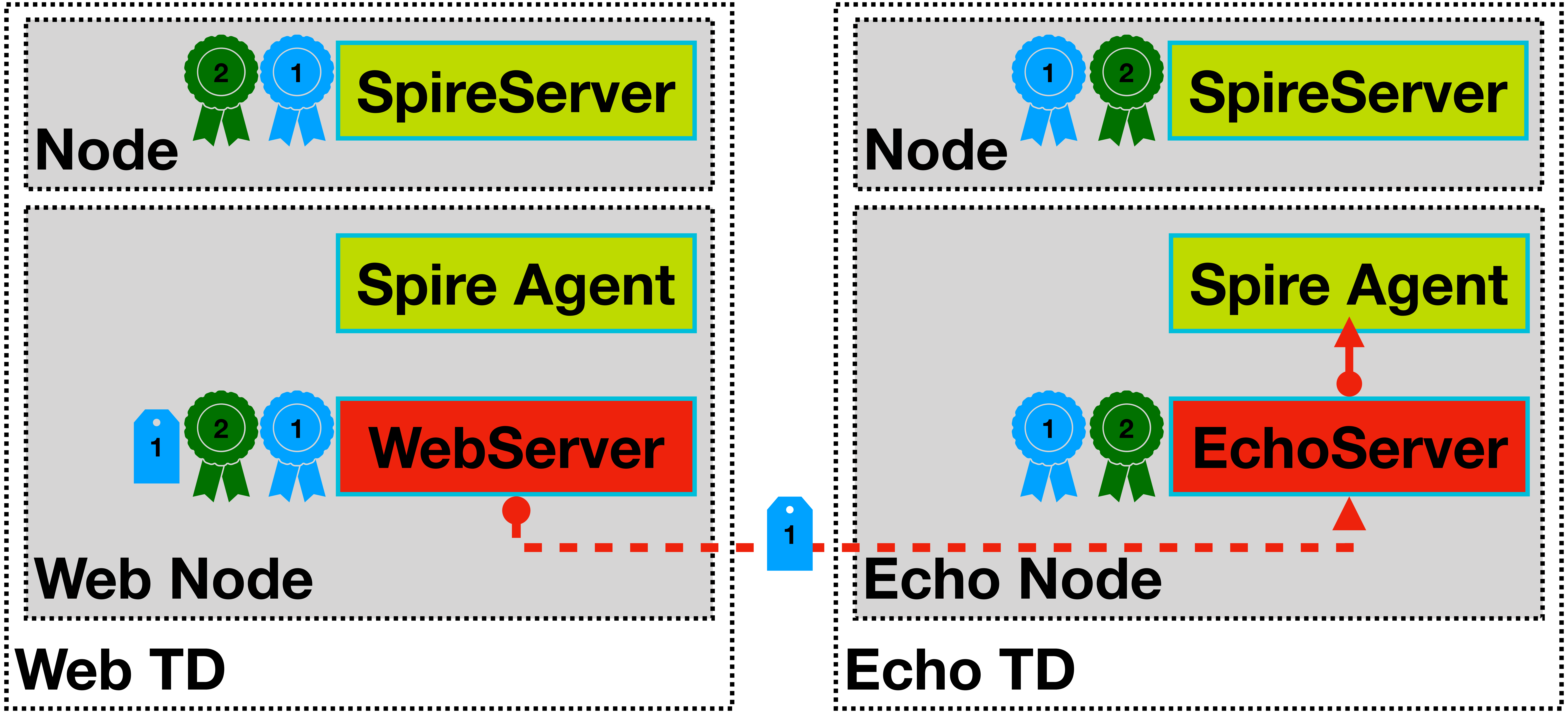

SpireServer
Node
Spire Agent
WebServer
Web Node
Web TD
SpireServer
Node
Spire Agent
EchoServer
Echo Node
Echo TD

JWT Flow


SpireServer
Node
Spire Agent
WebServer
Web Node
Web TD
SpireServer
Node
Spire Agent
EchoServer
Echo Node
Echo TD
1
2

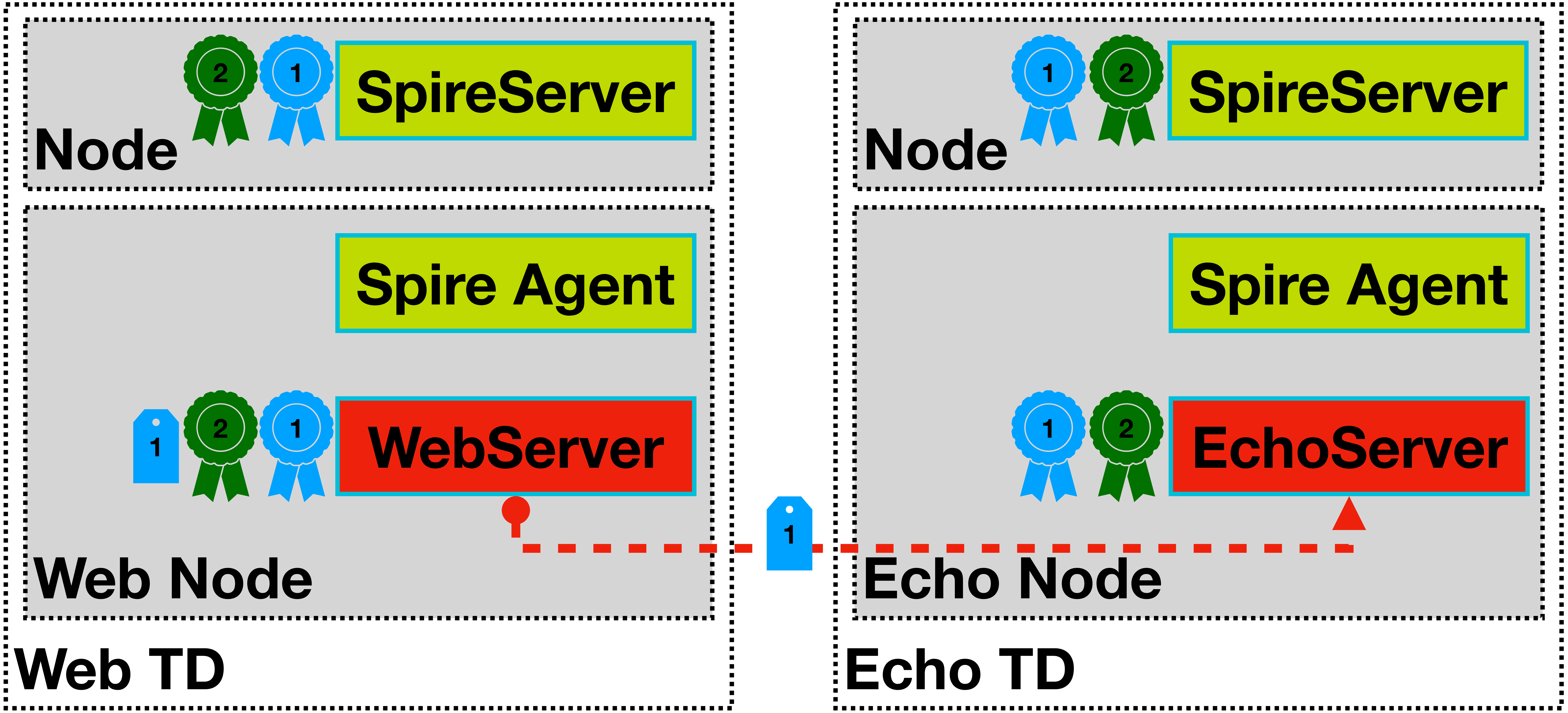
JWT Flow

ValidateJWTSVID Called



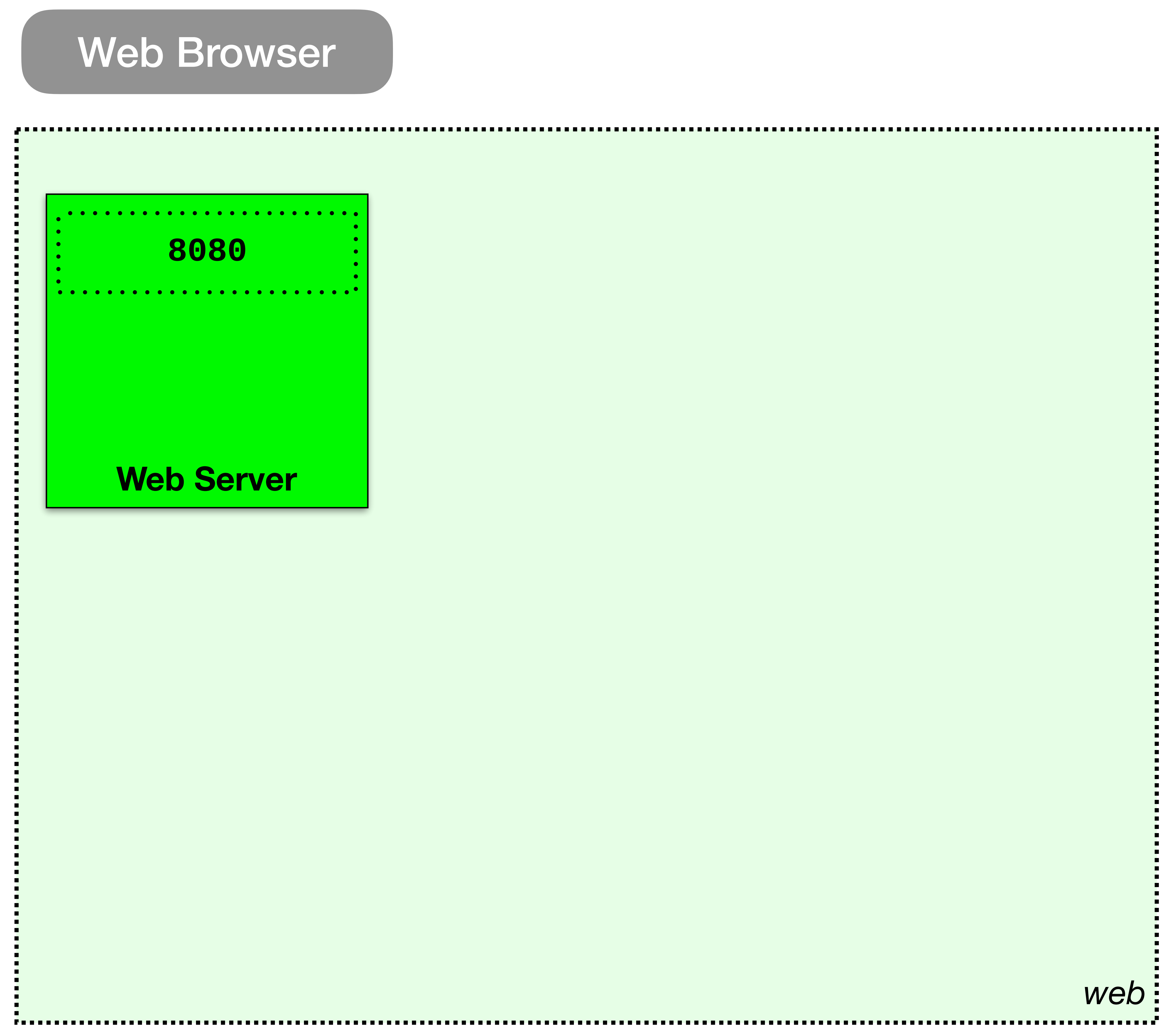
JWT Flow

YES

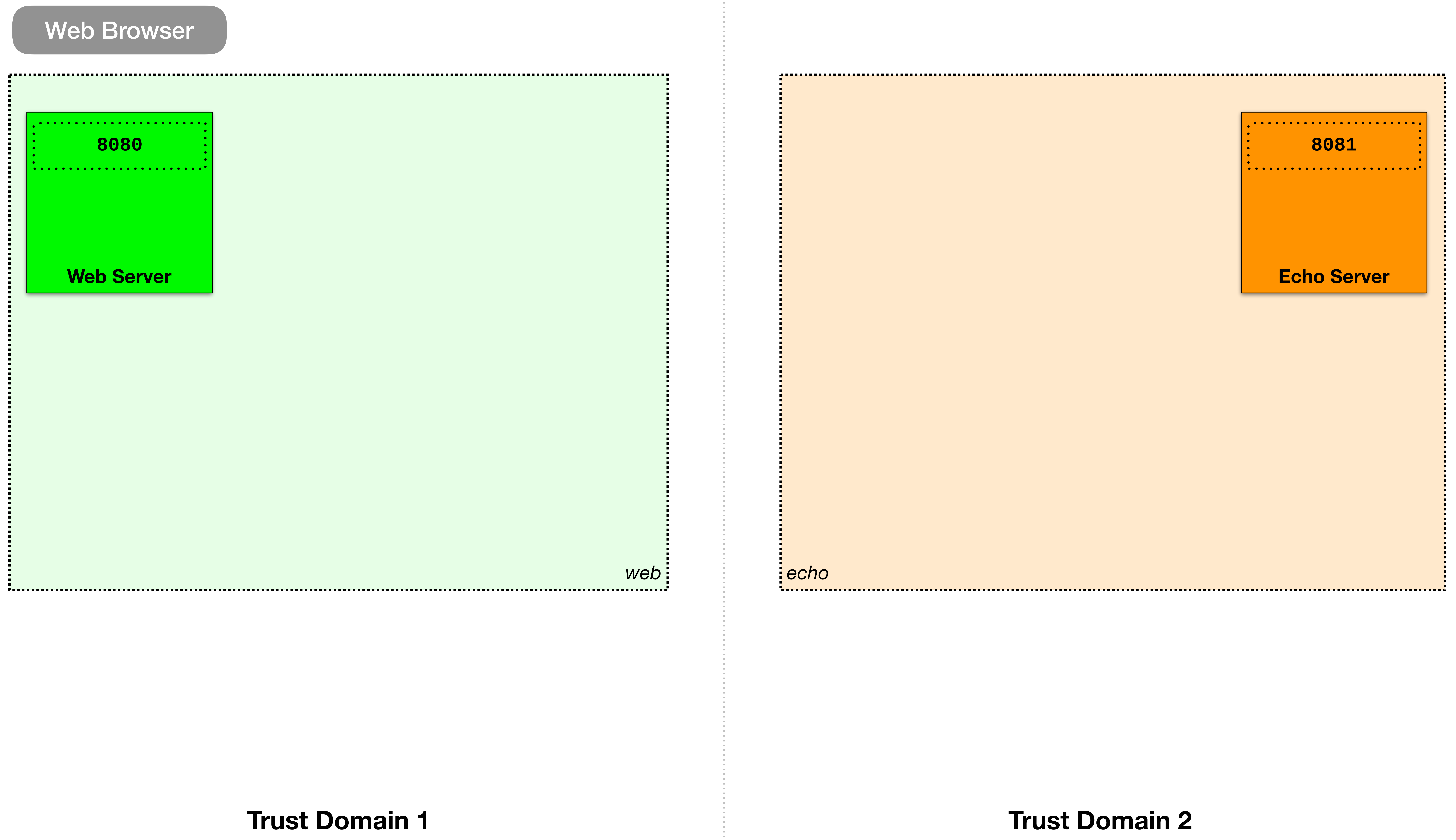

SpireServer
Node
SpireServer
Node
Spire Agent
Spire Agent
WebServer
EchoServer
Web Node
Echo Node
Web TD
Echo TD
1
2

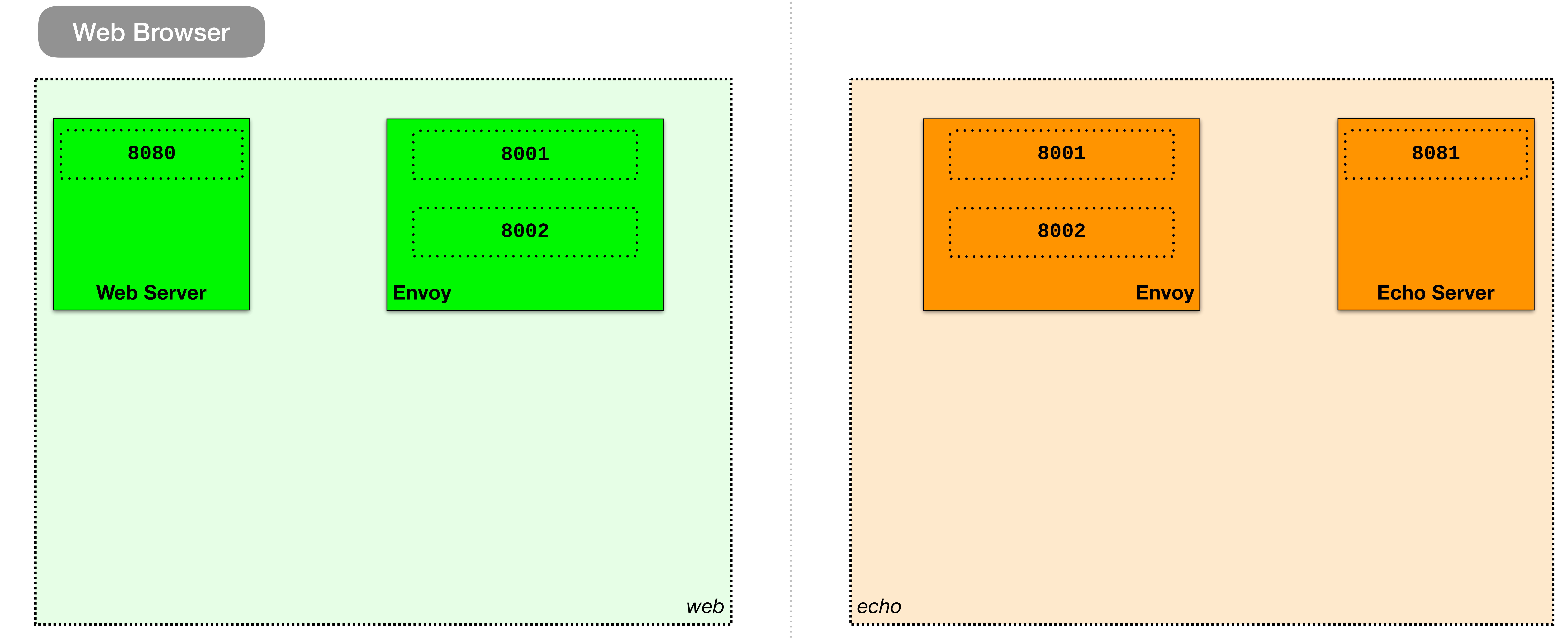
Demo

Web Browser



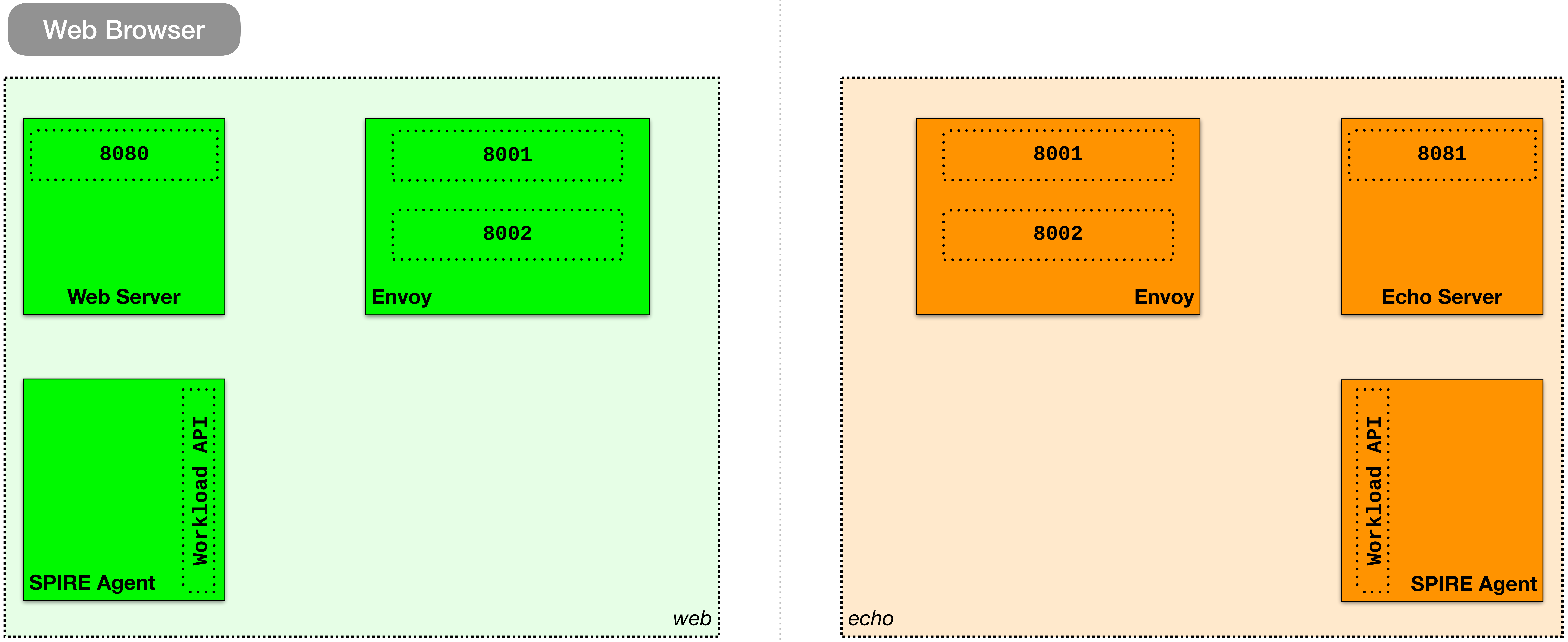
Trust Domain 1


Web Browser
8080
Web Server
web
8081
Echo Server
echo
Trust Domain 1
Trust Domain 2



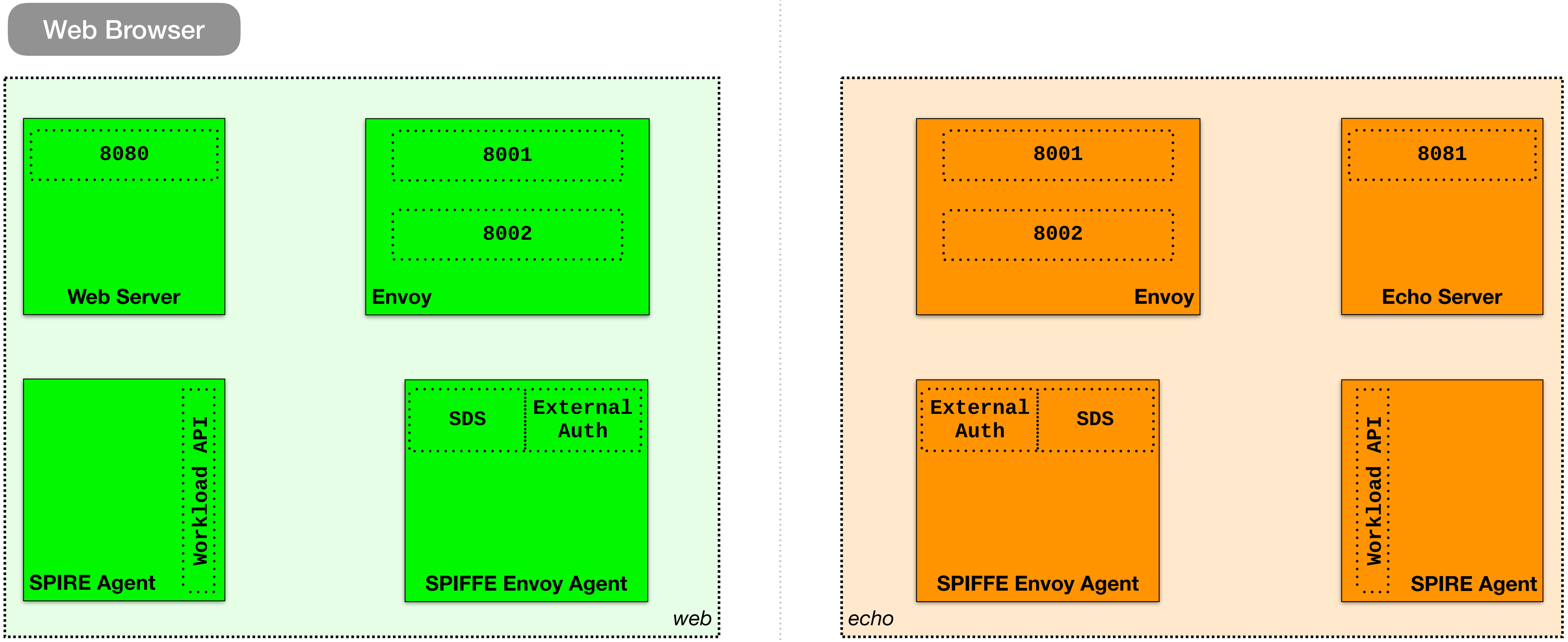
Trust Domain 1

Trust Domain 2



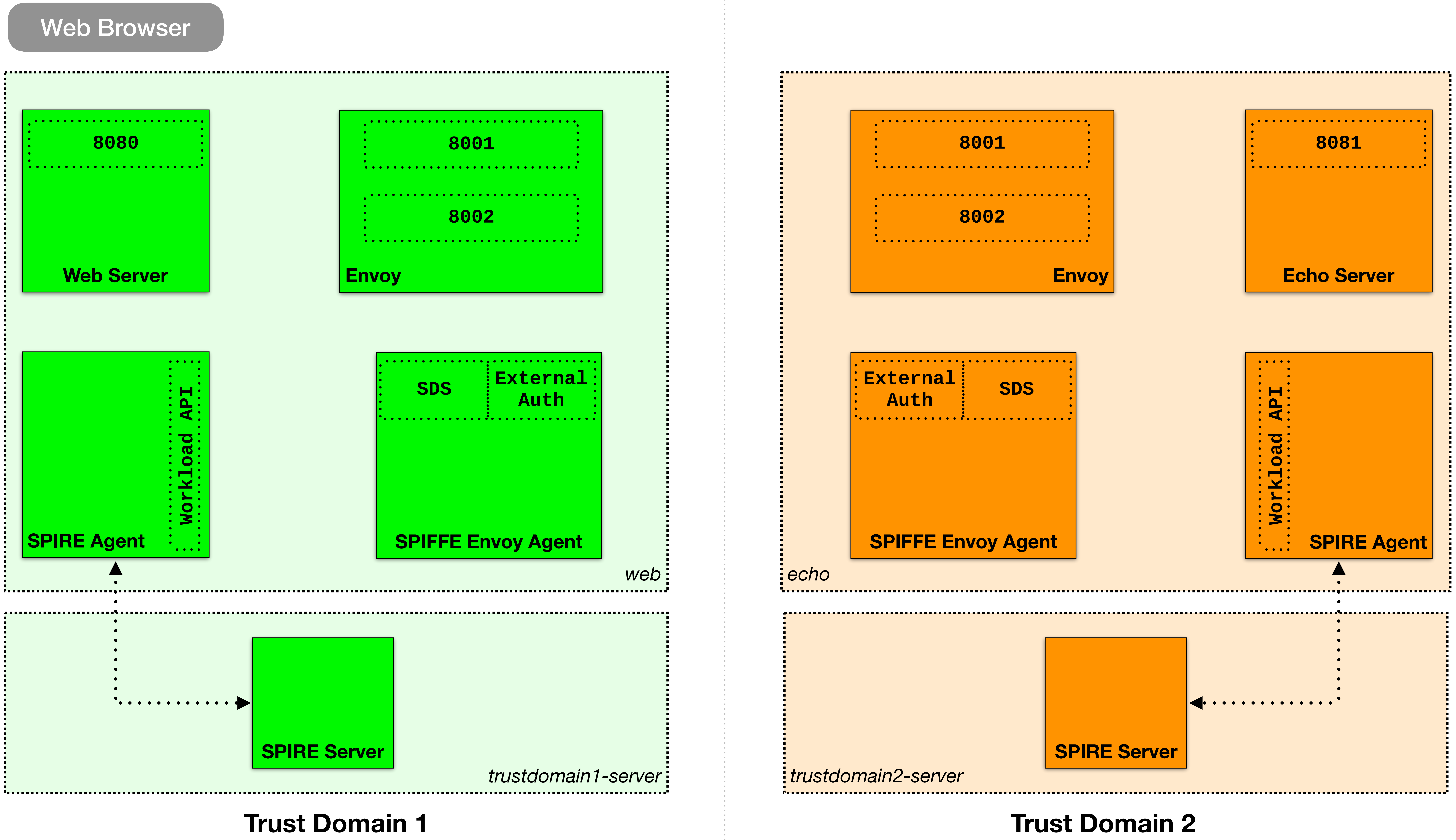
Trust Domain 1

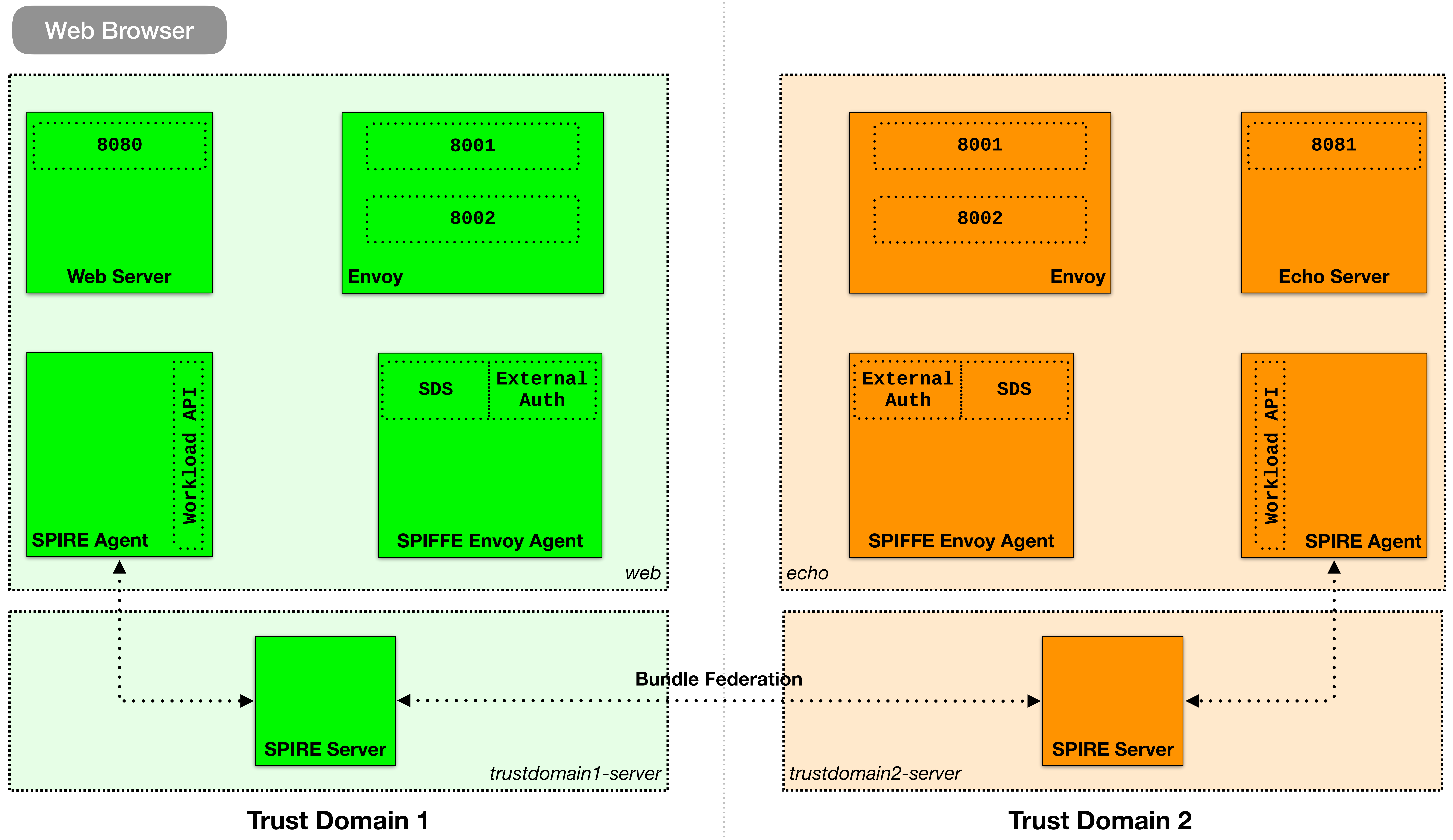
Trust Domain 2

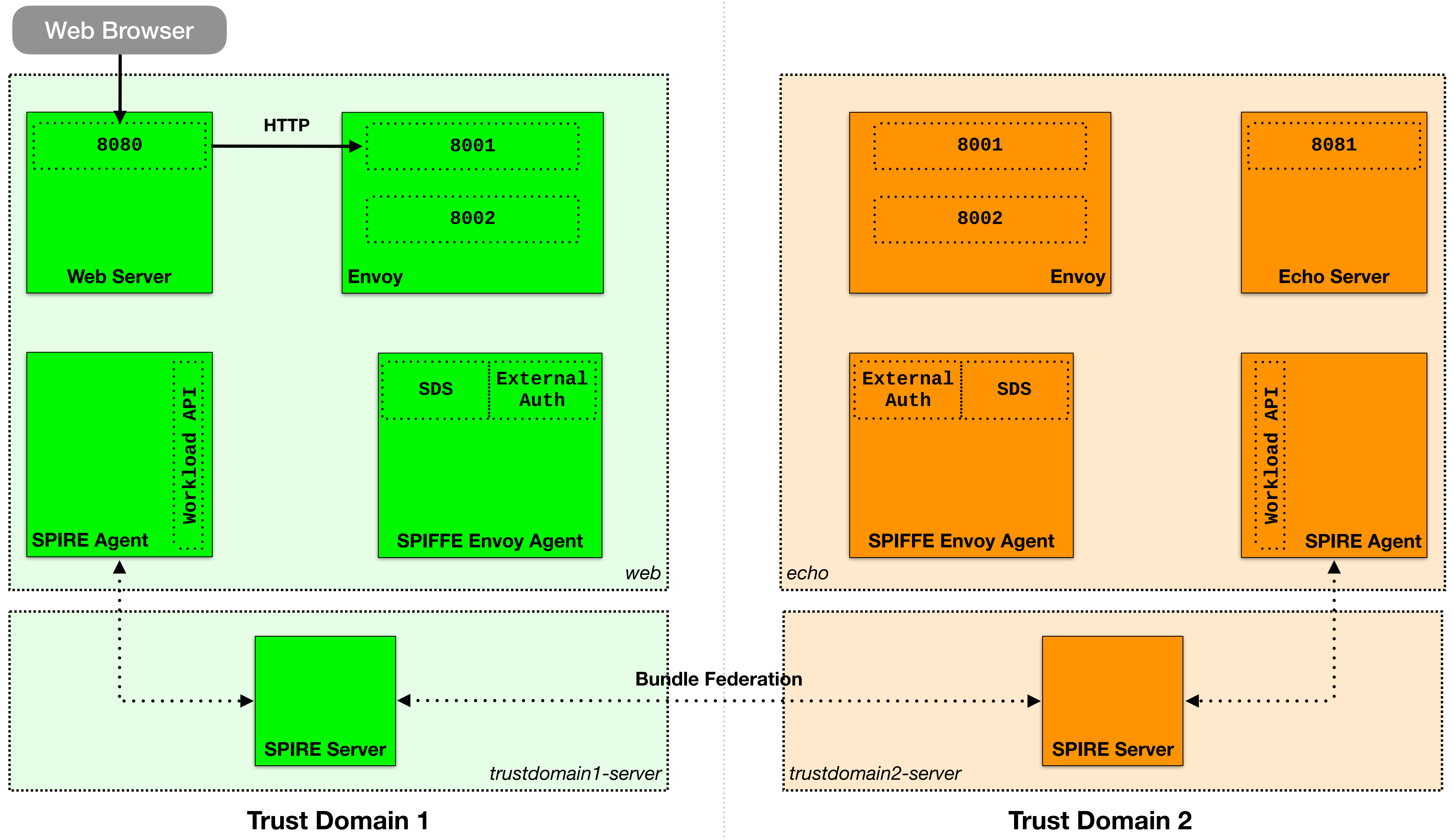


Trust Domain 1

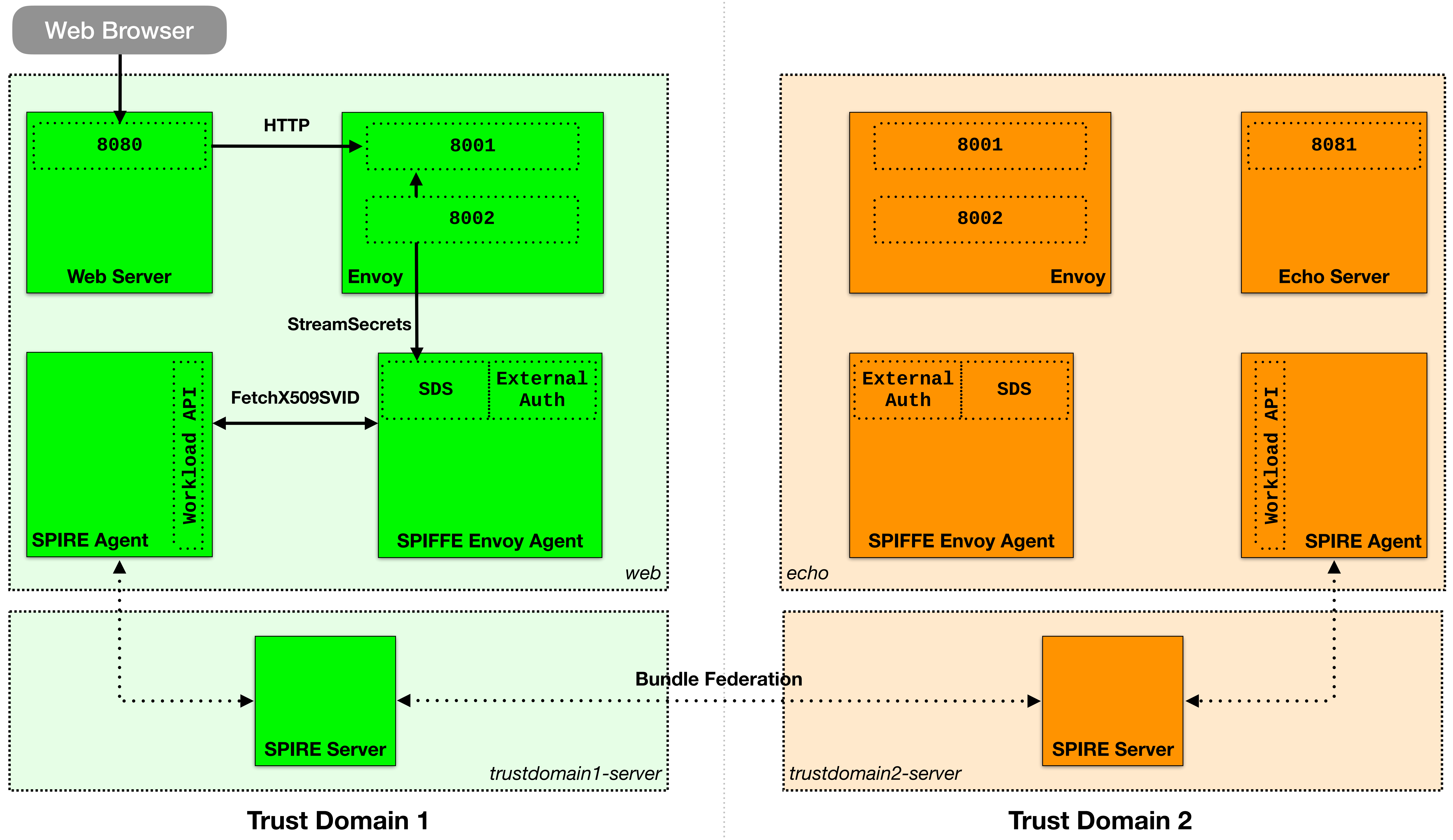
Trust Domain 2


Web Browser
8080
Web Server
8001
8002
Envoy
Workload API
SPIRE Agent
SDS
External Auth
SPIFFE Envoy Agent
web
SPIRE Server
trustdomain1-server
Trust Domain 1
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain2-server
Trust Domain 2

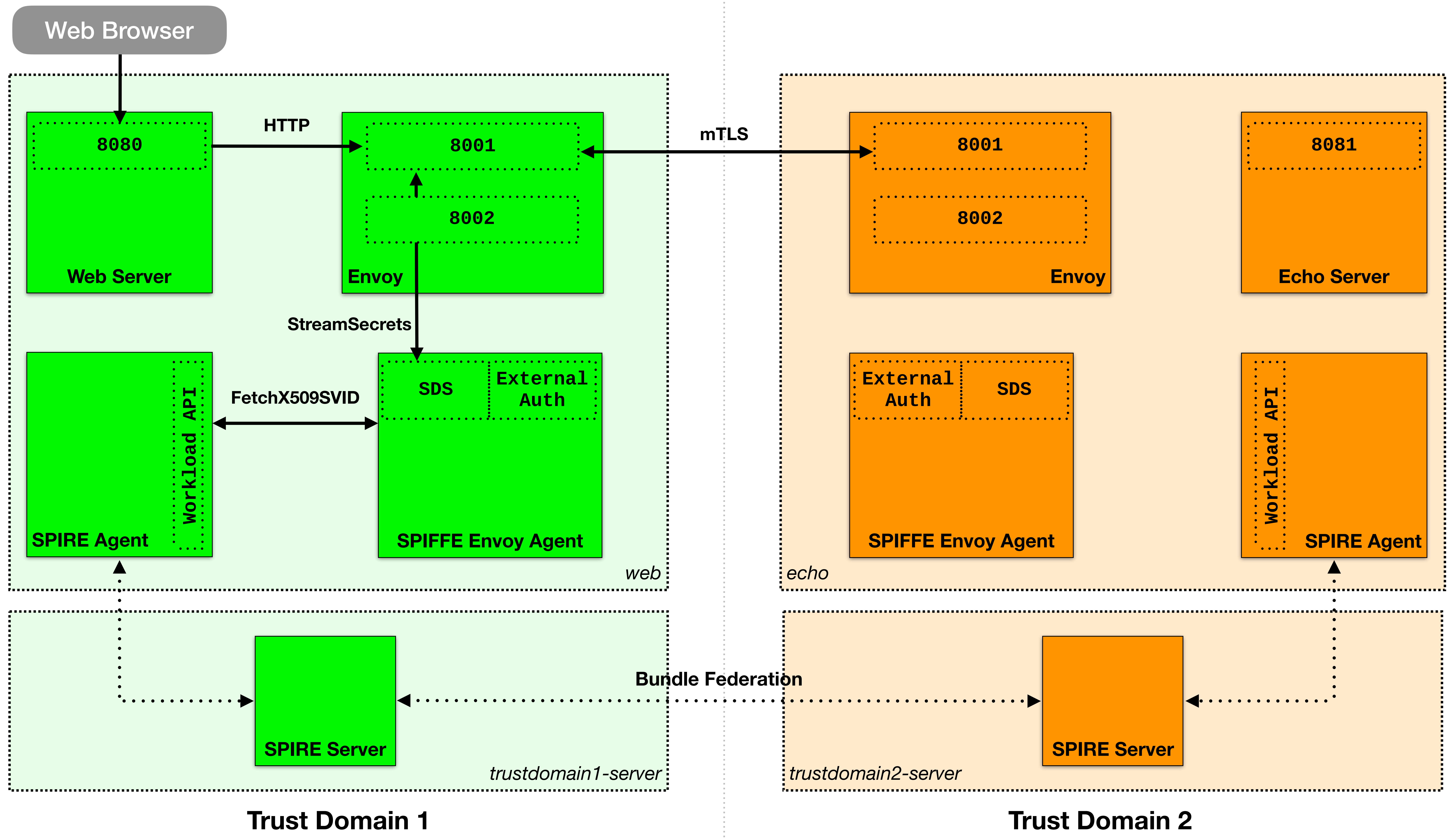

Web Browser
8080
Web Server
8001
8002
Envoy
SPIRE Agent
Workload API
SDS
External Auth
SPIFFE Envoy Agent
web
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain1-server
Bundle Federation
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2

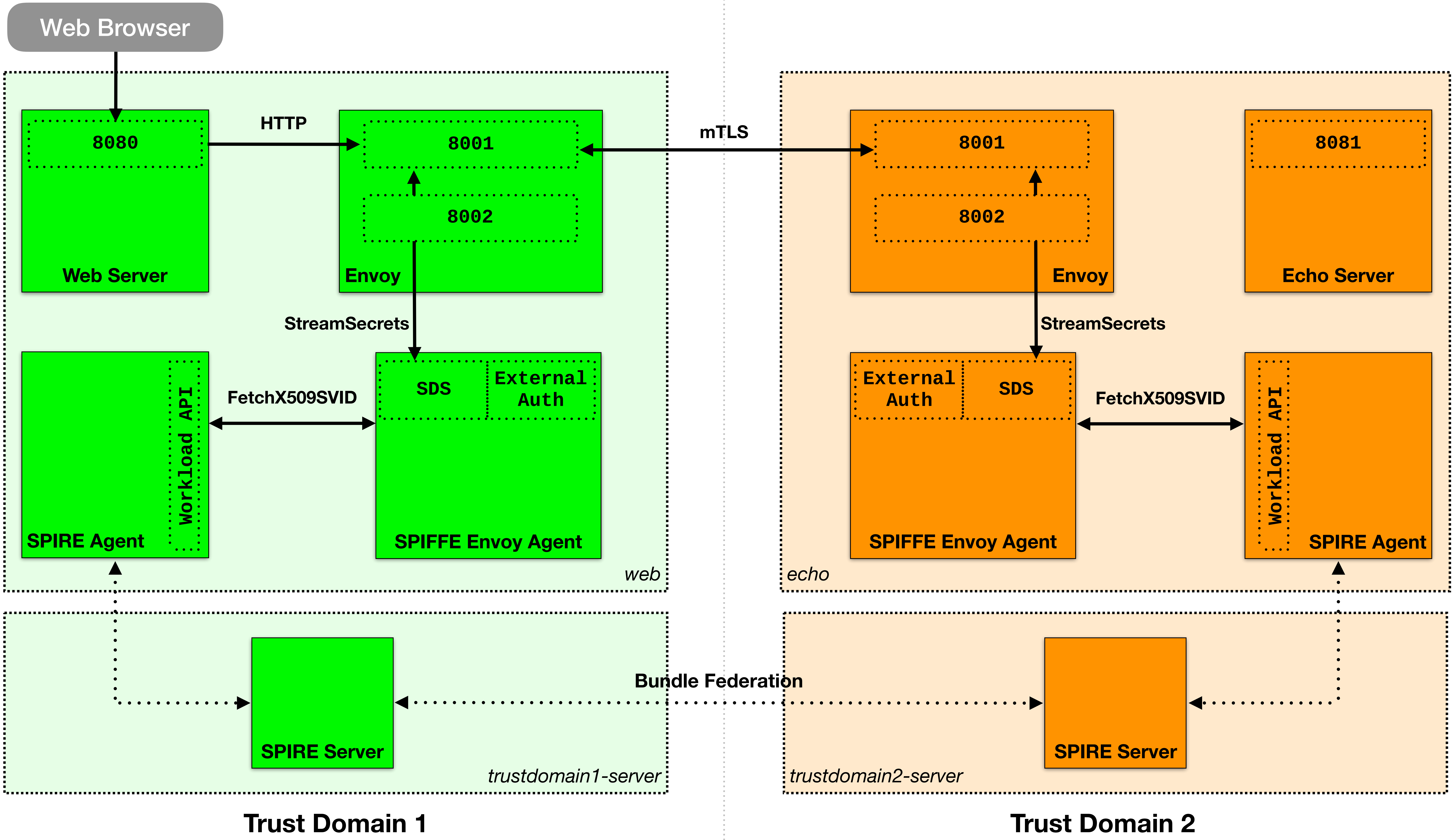


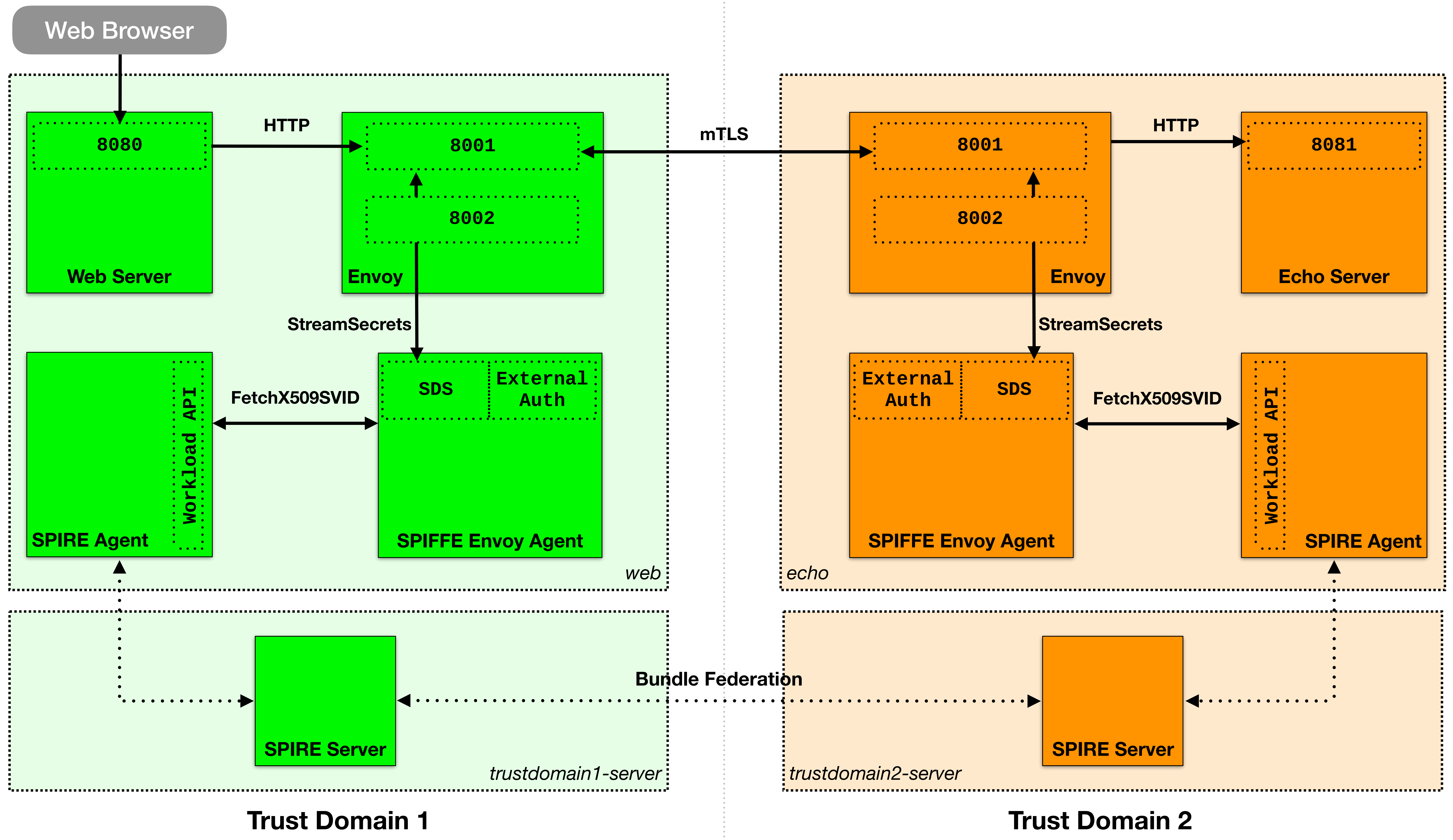
Web Browser
HTTP
8080
Web Server
8001
8002
Envoy
SPIRE Agent
Workload API
SDS
External Auth
SPIFFE Envoy Agent
web
SPIRE Server
trustdomain1-server
Bundle Federation
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2

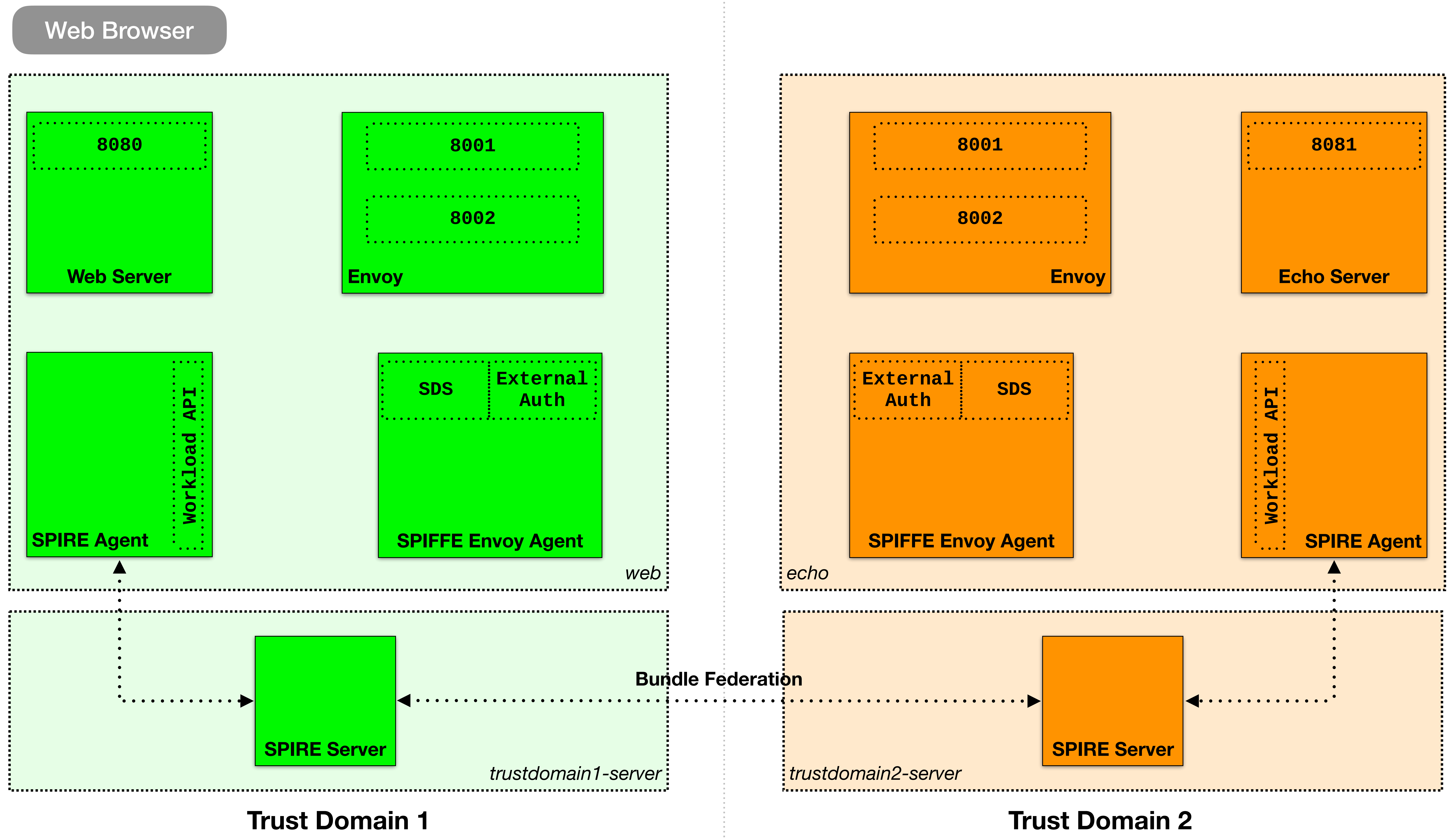


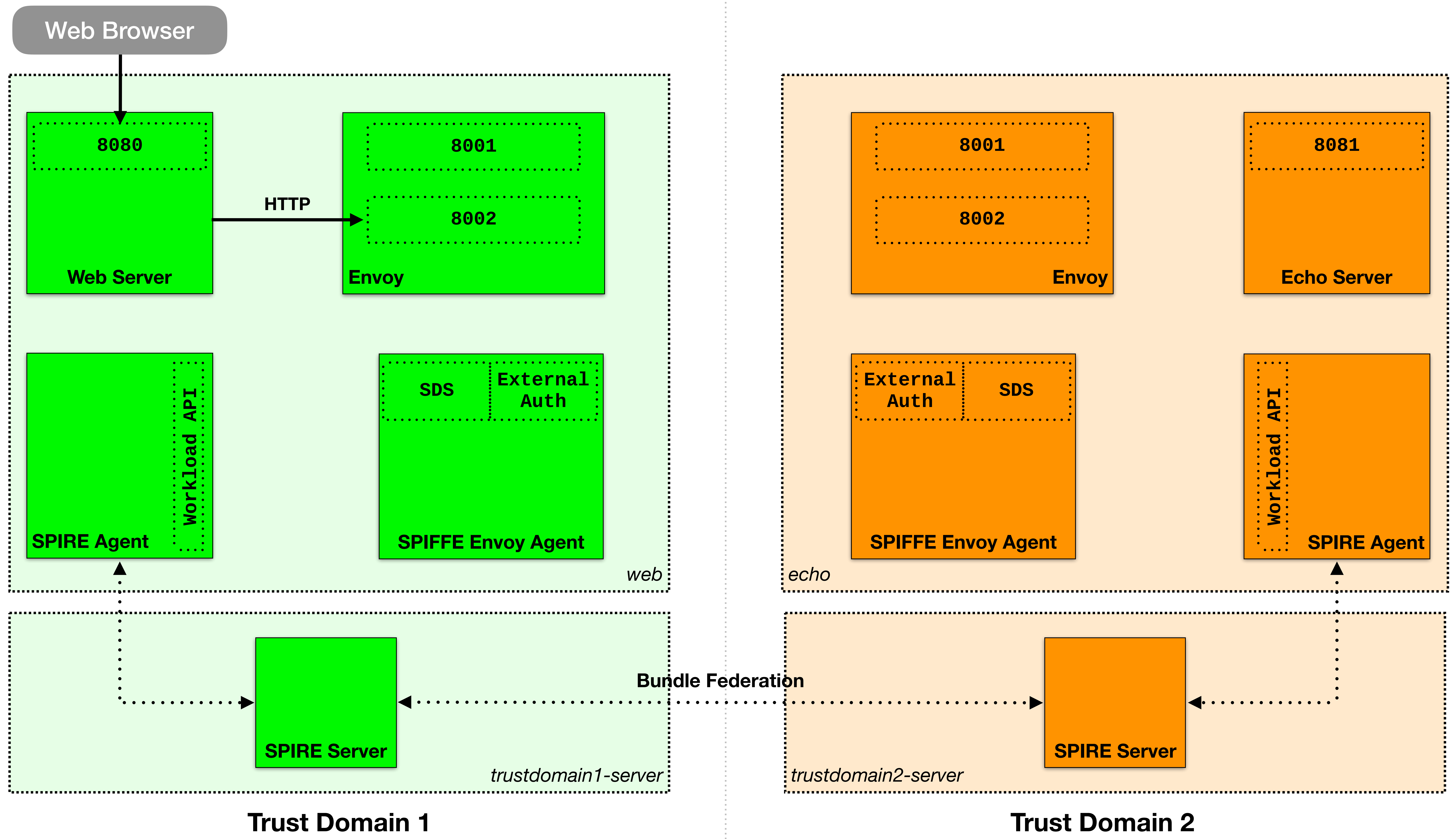
Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
StreamSecrets
SPIRE Agent
Workload API
FetchX509SVID
SDS
External Auth
SPIFFE Envoy Agent
web
SPIRE Server
trustdomain1-server
Bundle Federation
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2

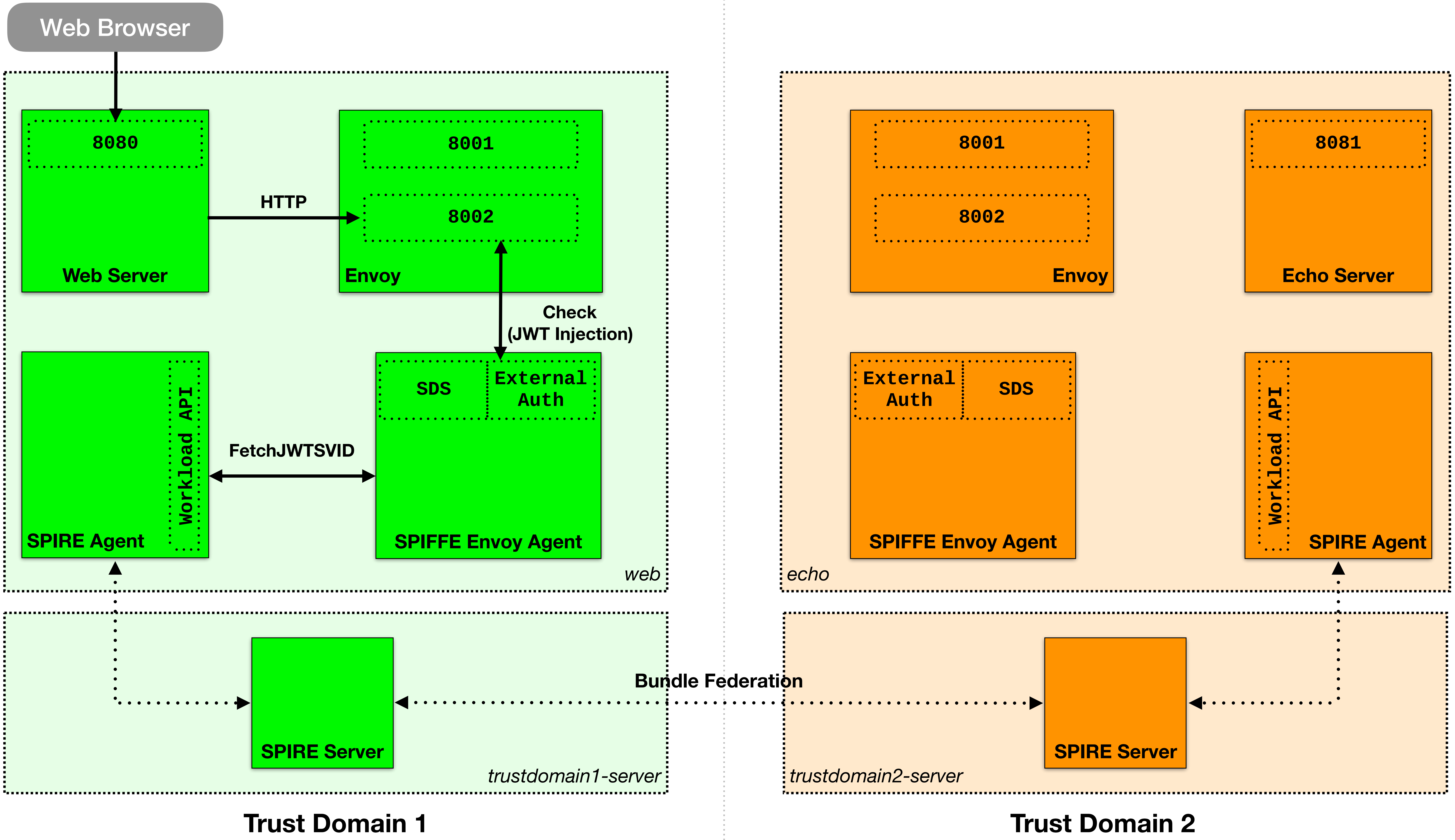

Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
mTLS
8001
8002
Envoy
8081
Echo Server
StreamSecrets
Workload API
SPIRE Agent
FetchX509SVID
SDS
External Auth
SPIFFE Envoy Agent
web
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
Bundle Federation
SPIRE Server
trustdomain1-server
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2

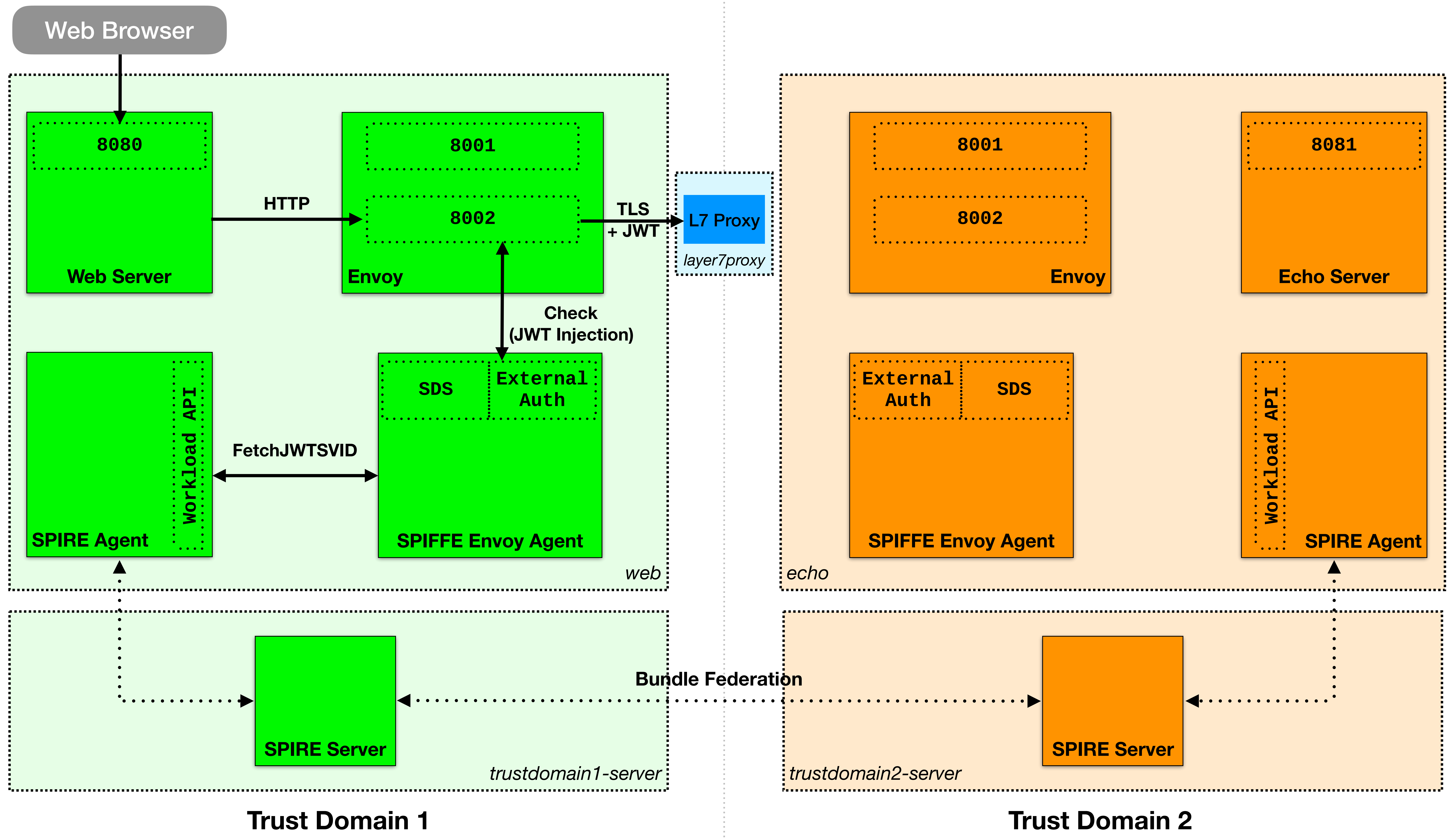

Web Browser
HTTP
8080
Web Server
8001
8002
Envoy
mTLS
8001
8002
Envoy
8081
Echo Server
StreamSecrets
StreamSecrets
SPIRE Agent
Workload API
FetchX509SVID
SDS
External Auth
SPIFFE Envoy Agent
External Auth
SDS
SPIFFE Envoy Agent
FetchX509SVID
Workload API
SPIRE Agent
web
echo
Bundle Federation
SPIRE Server
SPIRE Server
trustdomain1-server
trustdomain2-server
Trust Domain 1
Trust Domain 2

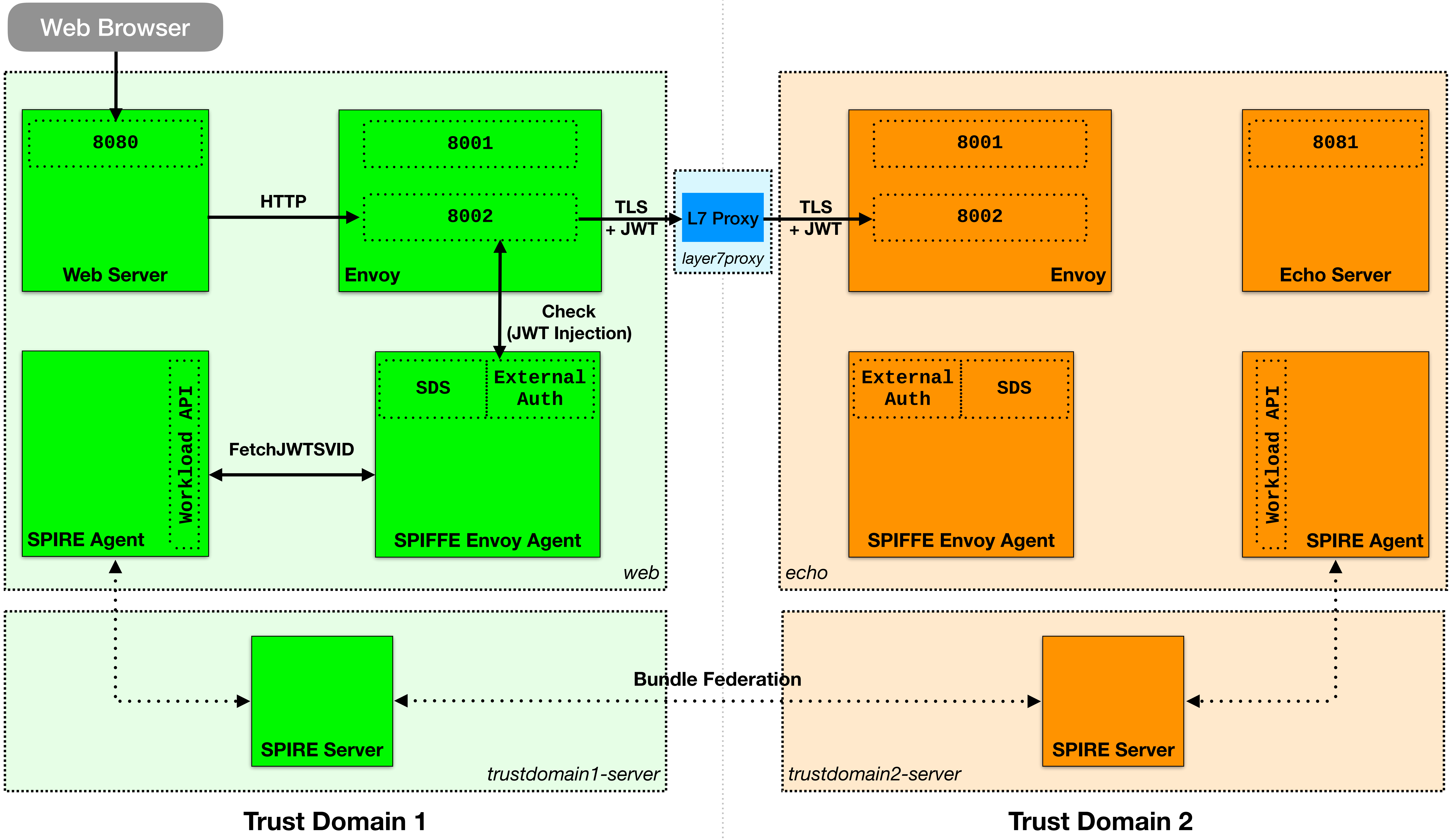

Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
mTLS
8001
8002
Envoy
HTTP
8081
Echo Server
StreamSecrets
StreamSecrets
Workload API
SPIRE Agent
FetchX509SVID
SDS
External Auth
SPIFFE Envoy Agent
web
External Auth
SDS
SPIFFE Envoy Agent
FetchX509SVID
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain1-server
Bundle Federation
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2


Web Browser
8080
Web Server
8001
8002
Envoy
Workload API
SPIRE Agent
SDS
External Auth
SPIFFE Envoy Agent
web
SPIRE Server
trustdomain1-server
Trust Domain 1
Bundle Federation
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain2-server
Trust Domain 2

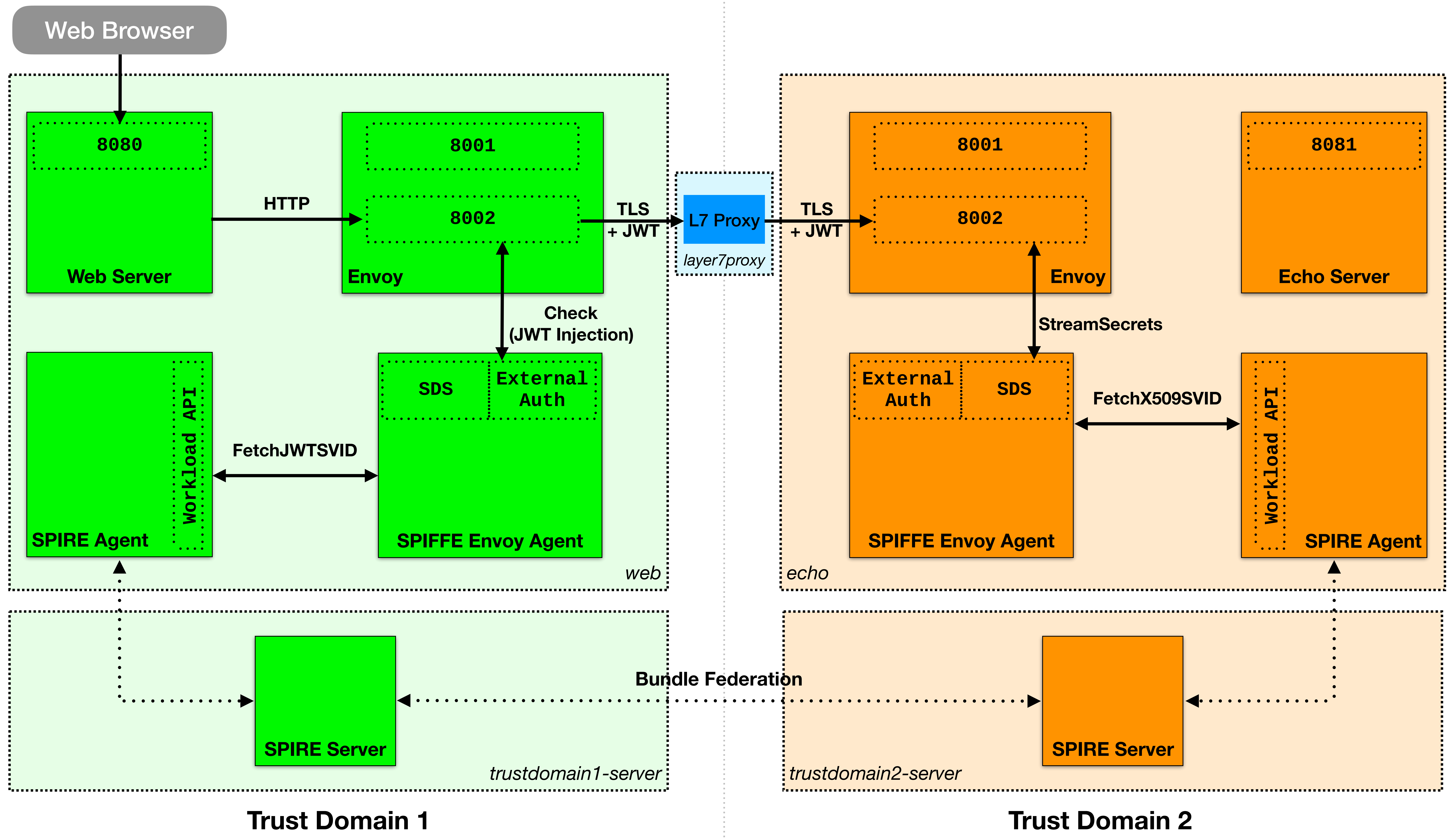

Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
Workload API
SPIRE Agent
SDS
External Auth
SPIFFE Envoy Agent
web
SPIRE Server
trustdomain1-server
Trust Domain 1
Bundle Federation
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain2-server
Trust Domain 2

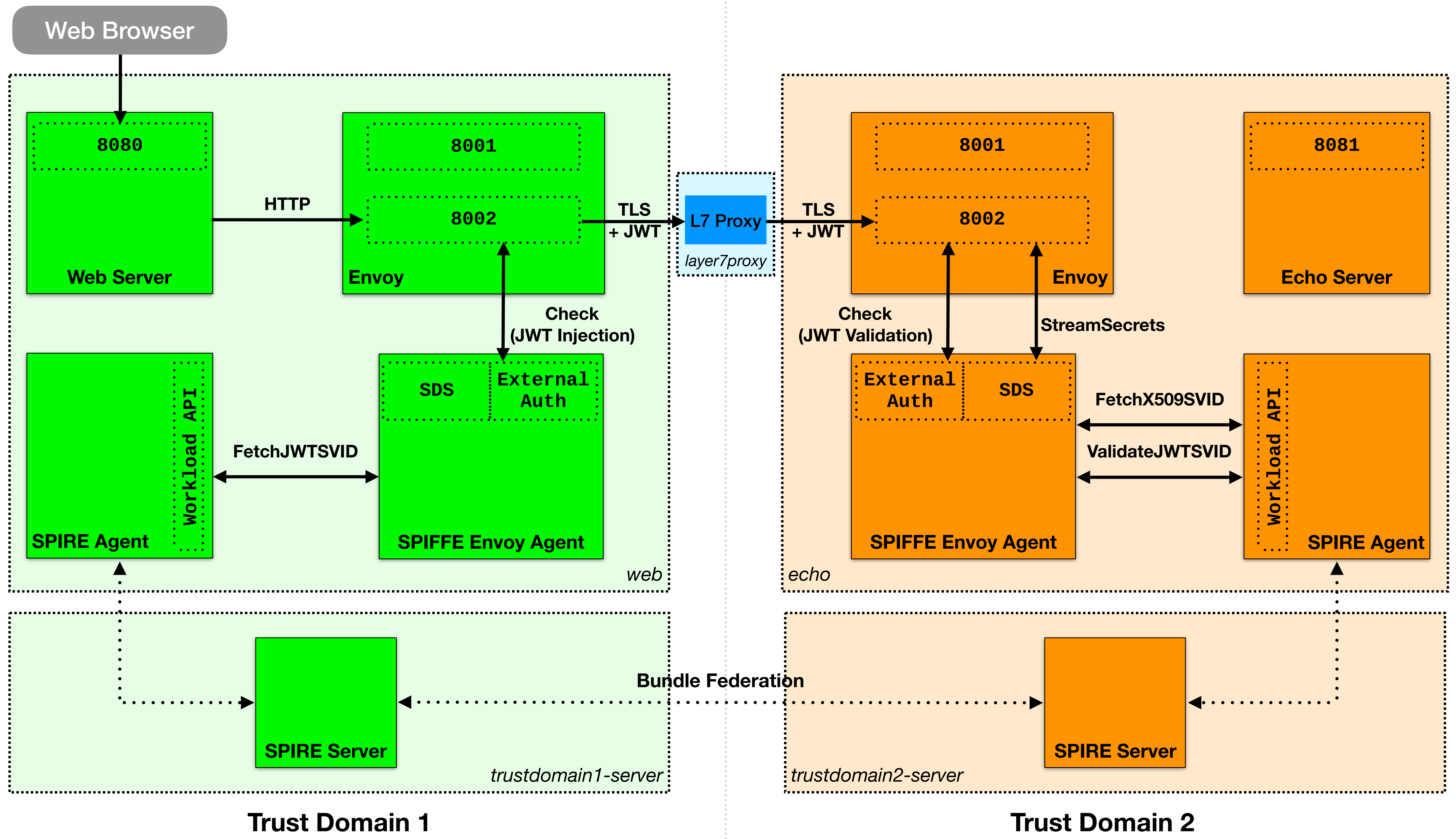

Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
Check
(JWT Injection)
SDS
External Auth
SPIRE Agent
Workload API
FetchJWTSVID
SPIFFE Envoy Agent
web
SPIRE Server
trustdomain1-server
Bundle Federation
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2


Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
TLS + JWT
L7 Proxy
layer7proxy
Check (JWT Injection)
SPIRE Agent
Workload API
FetchJWTSVID
SDS
External Auth
SPIFFE Envoy Agent
web
8001
8002
Envoy
8081
Echo Server
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
SPIRE Server
trustdomain1-server
Bundle Federation
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2

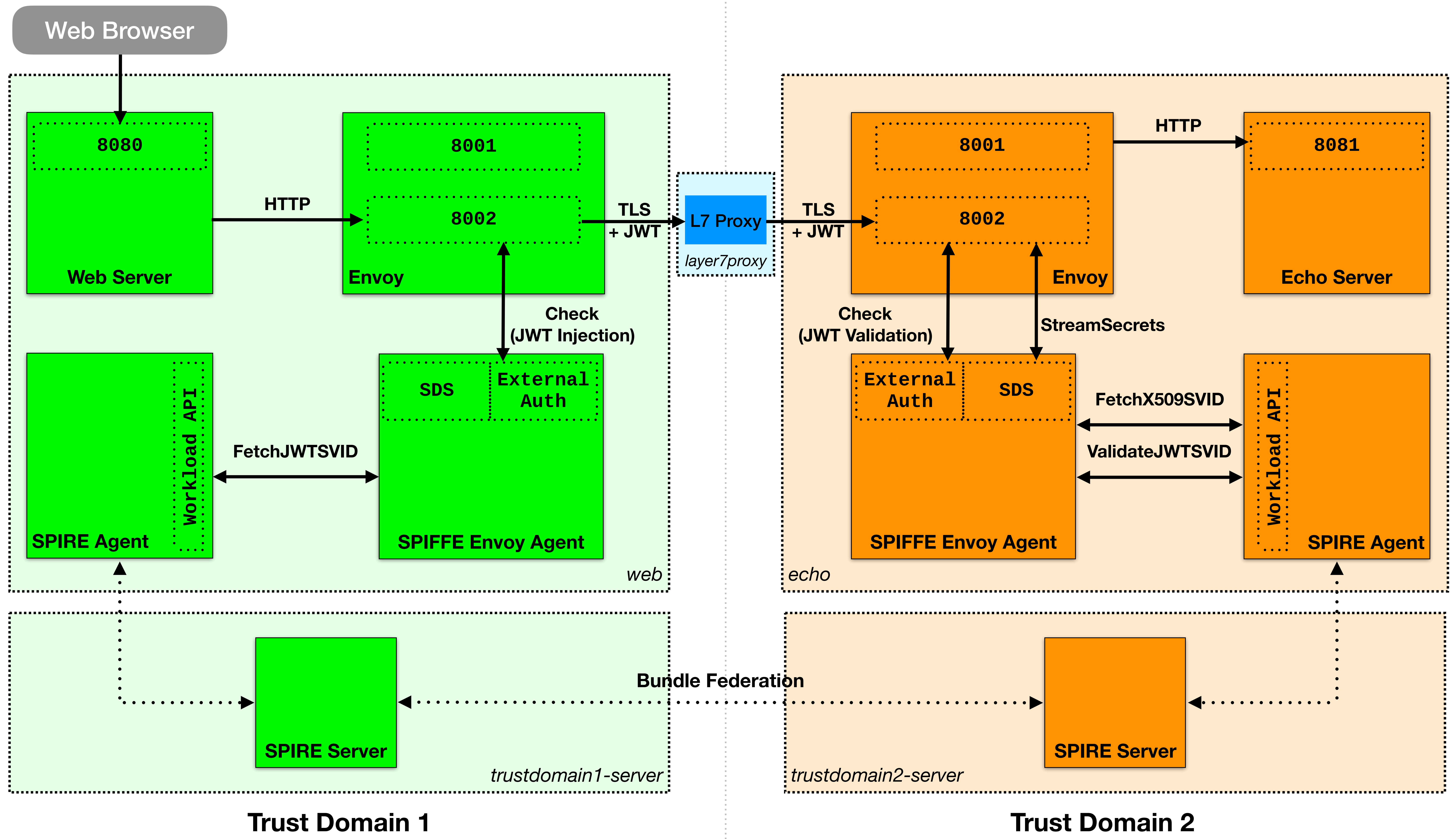


Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
TLS + JWT
L7 Proxy
layer7proxy
TLS + JWT
8001
8002
Envoy
8081
Echo Server
Check (JWT Injection)
SDS
External Auth
SPIFFE Envoy Agent
Workload API
SPIRE Agent
FetchJWTSVID
web
External Auth
SDS
SPIFFE Envoy Agent
Workload API
SPIRE Agent
echo
Bundle Federation
SPIRE Server
trustdomain1-server
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2


Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
TLS + JWT
L7 Proxy
layer7proxy
TLS + JWT
8001
8002
Envoy
8081
Echo Server
Check (JWT Injection)
StreamSecrets
SDS
External Auth
Workload API
SPIRE Agent
FetchJWTSVID
SPIFFE Envoy Agent
External Auth
SDS
FetchX509SVID
Workload API
SPIFFE Envoy Agent
SPIRE Agent
web
echo
Bundle Federation
SPIRE Server
trustdomain1-server
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2



Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
TLS + JWT
L7 Proxy
layer7proxy
TLS + JWT
8001
8002
Envoy
8081
Echo Server
Check (JWT Injection)
Check (JWT Validation)
StreamSecrets
SPIRE Agent
Workload API
FetchJWTSVID
SDS
External Auth
SPIFFE Envoy Agent
web
External Auth
SDS
SPIFFE Envoy Agent
FetchX509SVID
ValidateJWTSVID
Workload API
SPIRE Agent
echo
SPIRE Server
Bundle Federation
SPIRE Server
trustdomain1-server
trustdomain2-server
Trust Domain 1
Trust Domain 2


Web Browser
8080
Web Server
HTTP
8001
8002
Envoy
TLS + JWT
L7 Proxy
layer7proxy
TLS + JWT
8001
8002
Envoy
HTTP
8081
Echo Server
Check (JWT Injection)
Check (JWT Validation)
StreamSecrets
SPIRE Agent
Workload API
FetchJWTSVID
SDS
External Auth
SPIFFE Envoy Agent
web
External Auth
SDS
SPIFFE Envoy Agent
FetchX509SVID
ValidateJWTSVID
Workload API
SPIRE Agent
echo
Bundle Federation
SPIRE Server
trustdomain1-server
SPIRE Server
trustdomain2-server
Trust Domain 1
Trust Domain 2

Let’s Begin…

Source Code

- SPIFFE Envoy Agent

https://github.com/spiffe/spiffe-envoy-agent

- Demo

https://github.com/spiffe/spiffe-example/spiffe-envoy-agent

What is next

- Federation API work in SIG-SPEC
- SPIFFE Bundle Format
- SPIRE Roadmap
- Implement Envoy support into SPIRE Agent

slack.spiffe.io

github.com/spiffe

spiffe.io

Questions?