

# Central Authentication System

## *Web Application* のための新しい認証システムの試み

内藤 久資

naito@math.nagoya-u.ac.jp

名古屋大学多元数理科学研究科

梶田将司氏（名古屋大学情報連携基盤センター）との共同研究

# はじめに

- CAS : Yale 大学が開発した Open Source software
  - 基本的な “Authentication” のみを提供
  - 強力な “Authorization” 機能を追加 (nu-CAS)
- 特徴
  - Web Application に対する認証に特化したシステム
  - Web Application 側には特権が必要ない
  - Backend Database の種類に依存しない
  - Authentication の一元化と Authorization の分散管理

# 認証 (AAA) とは何か

## ● Authentication ( 認証 )

- 「ユーザ」は正当なユーザか？
- User ID & Password
- Identification & Authentication (I&A)

## ● Authorization ( 認可 )

- 各リソースに対して  
「ユーザ」はアクセスする権利を持つか？
- Access Permission

## ● Accounting ( 課金 )

- 「ユーザ」が何をどれだけ利用したか？
- Logging

# Web Application の認証の必要性と方法

- Web ページに対するアクセス制限の必要性
  - 機関内部だけに公開されている Web ページ
    - IP アドレスによる制限
    - AAA は特に必要なし
  - 個人を特定したサービスの提供 (Portal サイト, Web Shopping, ...)
    - Apache Basic 認証, ...
    - Authentication が必要
    - Authorization, Session 管理 が必要な場合もある
  - システム管理のための CGI
    - Apache Basic 認証, ....
    - Authentication & Authorization が必要

# 従来の認証方法の問題点

- Basic 認証
  - 「パスワードファイル」を読み出せない
- Module による外部認証: `mod_radius`, `mod_ldap`, ...
  - Authentication のみが可能
  - Authorization, Session trucking が困難
- 電子証明書によるクライアント認証
  - Authentication, Authentication が可能
  - Session trucking が困難
  - クライアント証明書の入手が困難
- 全て, 個々の HTTPD server, directory レベルでの設定が必要

# CAS による認証の利点 (1: Yale CAS)

- Authentication そのものは外部認証サーバを利用
  - LDAP, NetInfo, Active Directory などの Directory Service
  - Radius, NIS, BSD Flat File などの外部データベース
- Authentication は CAS Server が行う
  - 個々の Web Application には「特権」は必要ない
- 簡単な Session 管理 (Session Timeout 管理) が可能
- ほとんどの Web Application で利用可能
  - Cookie, JavaScript
  - CAS client: C, Java, Perl, Ruby, PL/SQL, PHP,....
  - **mod\_cas, pam\_cas**

## CAS による認証の利点 (2: nu-CAS)

- Authorization にも, 外部データベースを利用
  - Authorization Data の分散管理が可能
- Authorization も CAS Server が行う
  - 「ユーザ」グループ, Remote Host, 時間帯制限, ....
  - Application に渡す Authentication データを制御可能
- Session 管理が可能
  - 二重ログインの防止
  - Session Trucking
    - Application に Cookie を渡すことで Application 側が実装可能

# CAS の構成

- JAVA 1.4 API, Servlet 2.3 API 上で動作
  - Tomcat 5.x 上での動作が確認されている
- ブラウザに対する要求事項
  - Cookie の利用が可能なこと
  - 極めて単純な JavaScript が動作すること



# CAS 認証のしくみ

## ● 用意すべきもの

- Web Application Server (including CAS client)
- CAS Server (over Tomcat)
- Directory Server (example LDAP Server)
- Web Browser

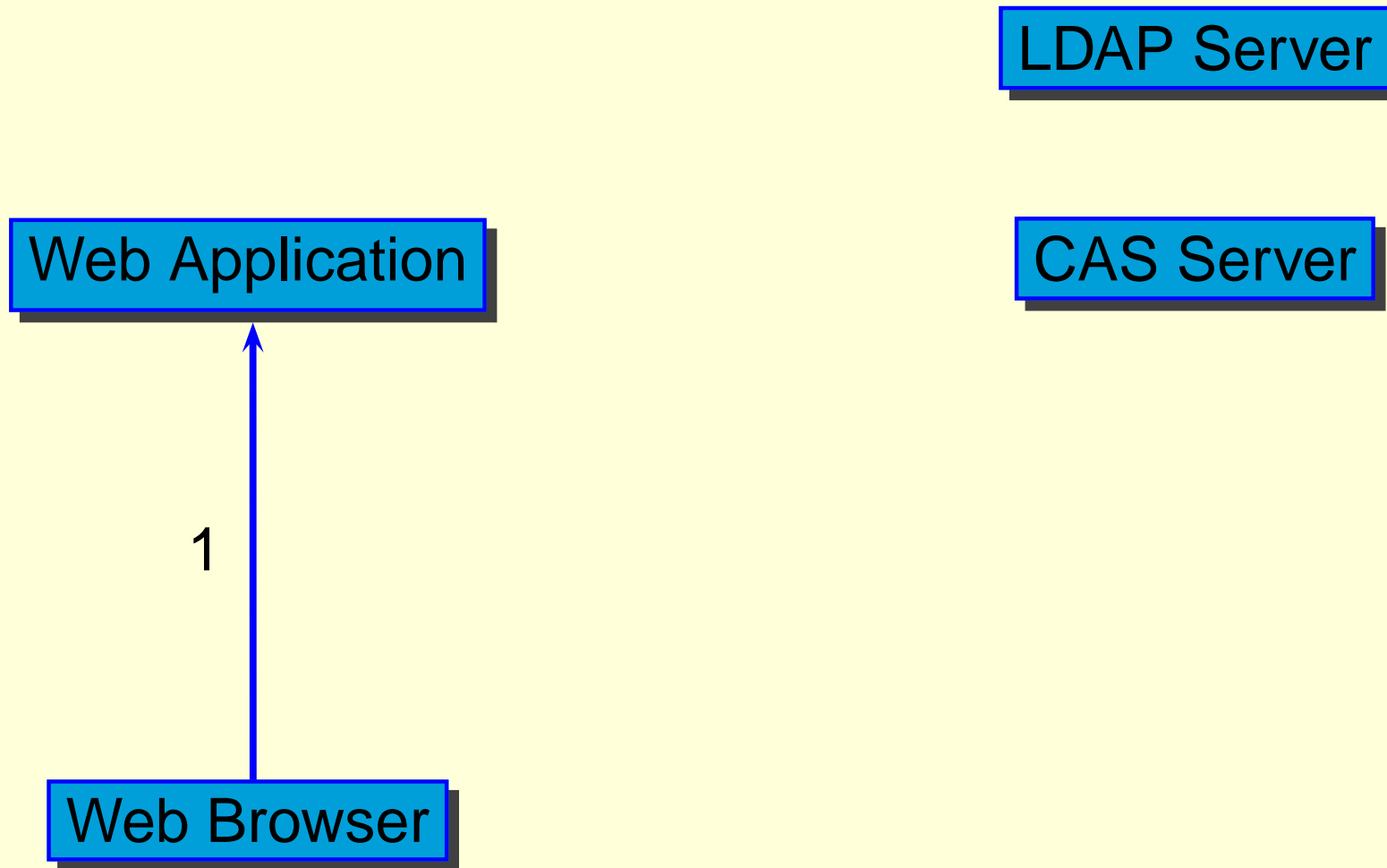
## ● 登場する概念

- Ticket Granting Cookie (TGC)
- Service Ticket (ST)

# CAS 認証のしくみ

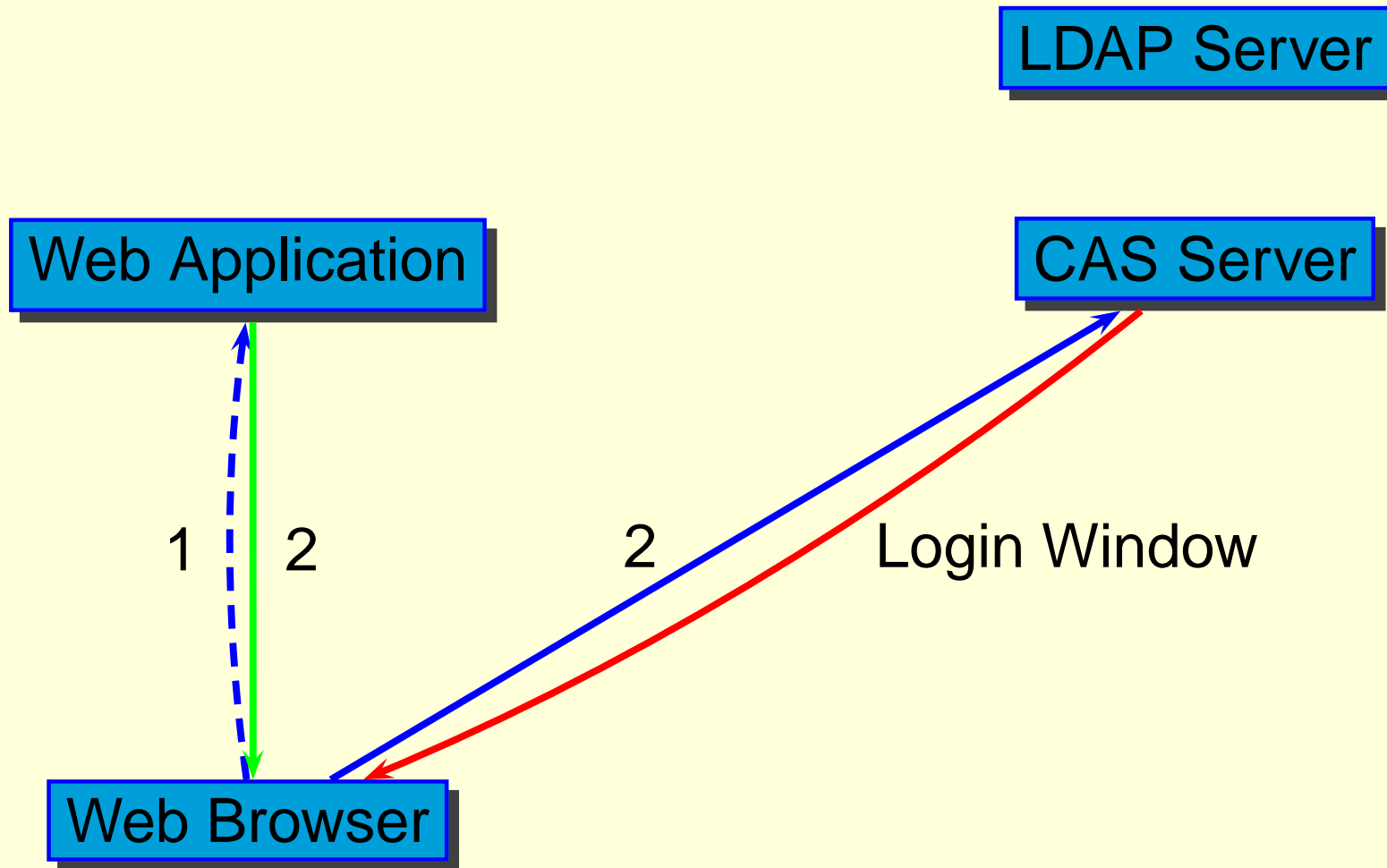
- Login しているユーザのブラウザには TGC が保存される
  - CAS Server は TGC & ST データベースを保存
- 1 回のアクセスごとに ST を発行
  - ST は One Time Ticket
  - ST は TGC に付随
- TGC で Authentication, ST で Authorization を行う
- ST Varidation      Application に「ユーザのデータ」を送信
- TGC の Timeout = Session Timeout
- TGC の削除 = Logout

# CAS 認証のしくみ (1: Login (1))



1. Access to <https://aFQDN/a.html>

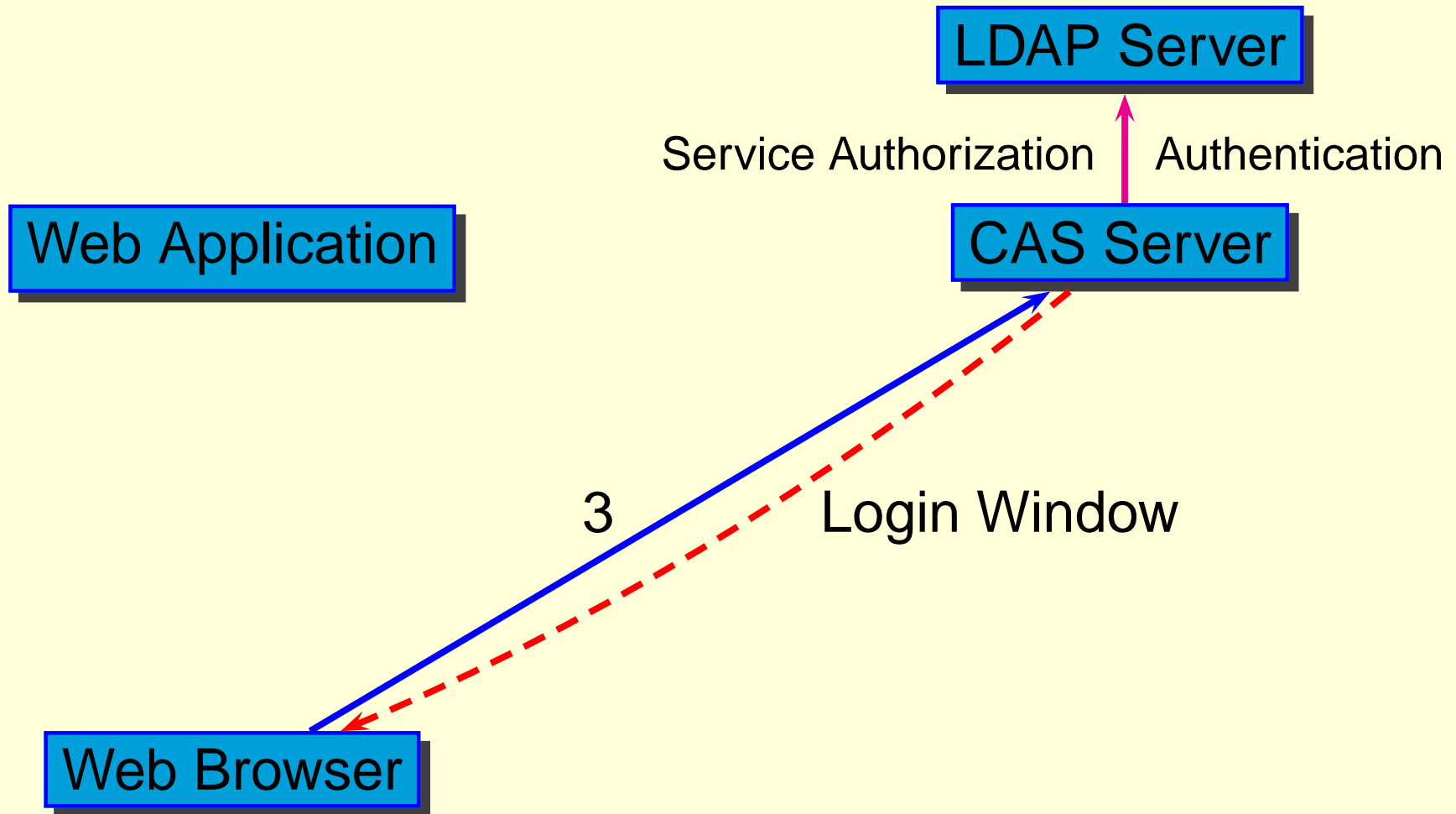
# CAS 認証のしくみ (1: Login (2))



2. Redirect to

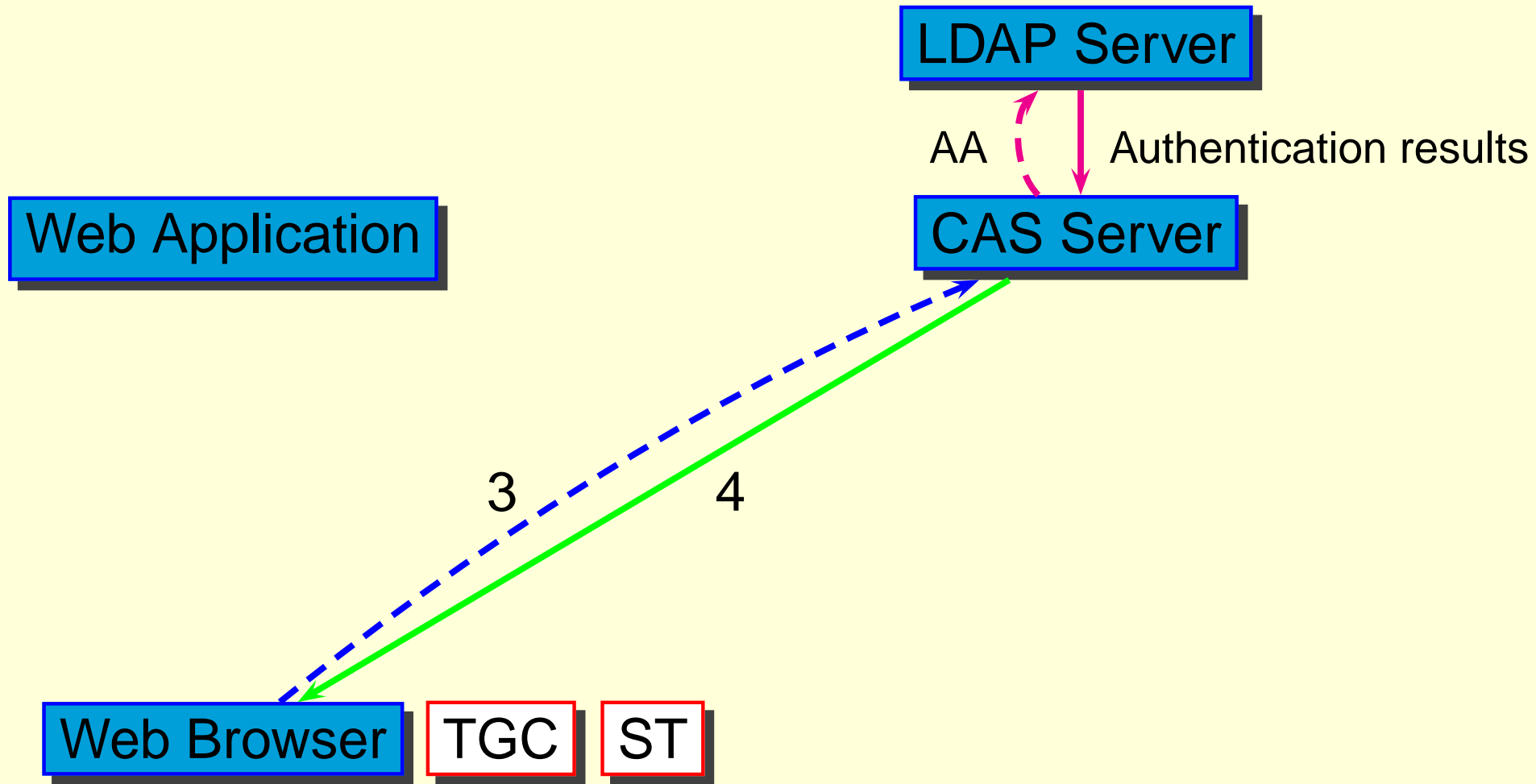
<https://CAS/login&service=https://aFQDN/a.html>

# CAS 認証のしくみ (1: Login (3))



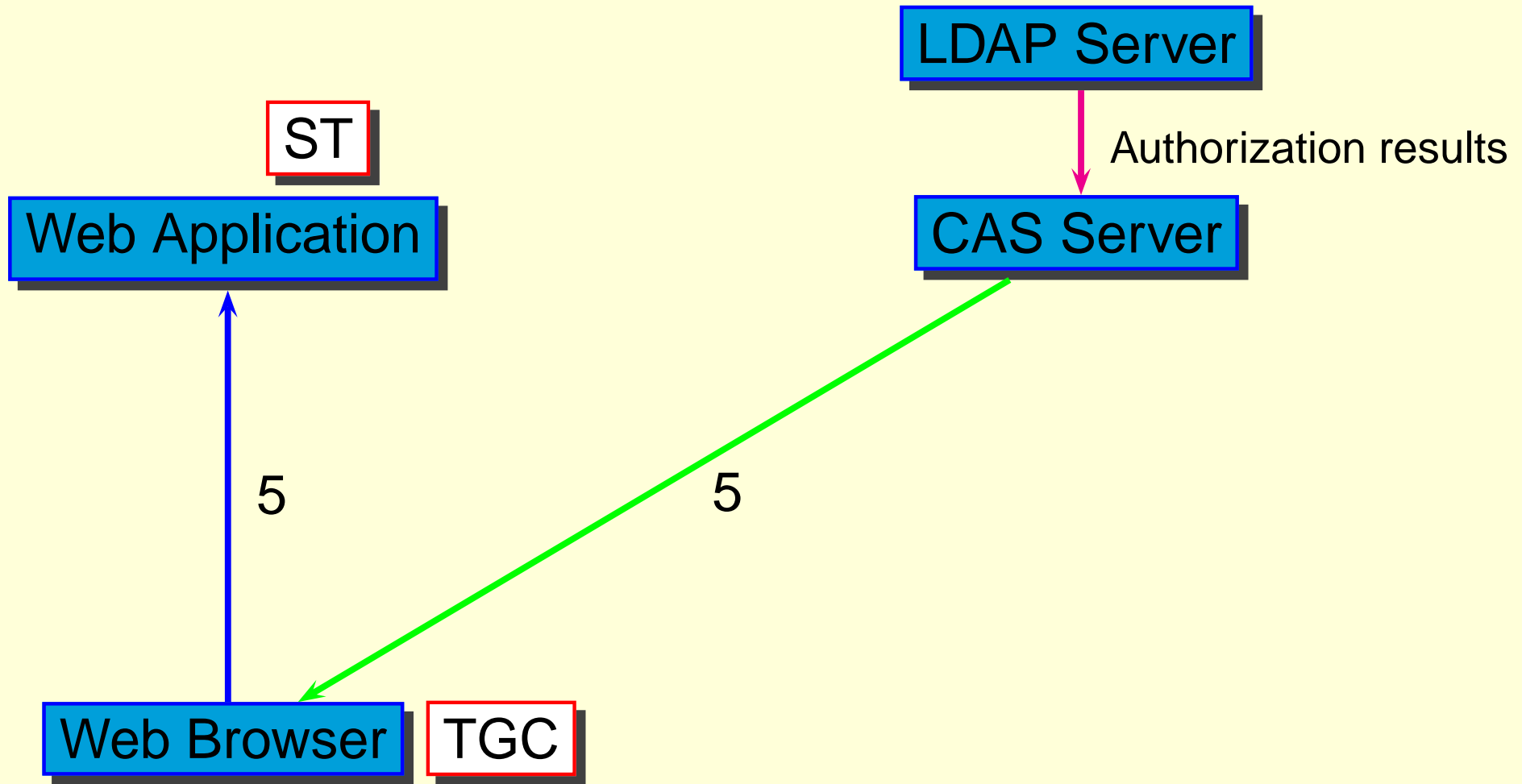
3. Input UserID & Password with service <https://aFQDN/a.html>

# CAS 認証のしくみ (1: Login (4))



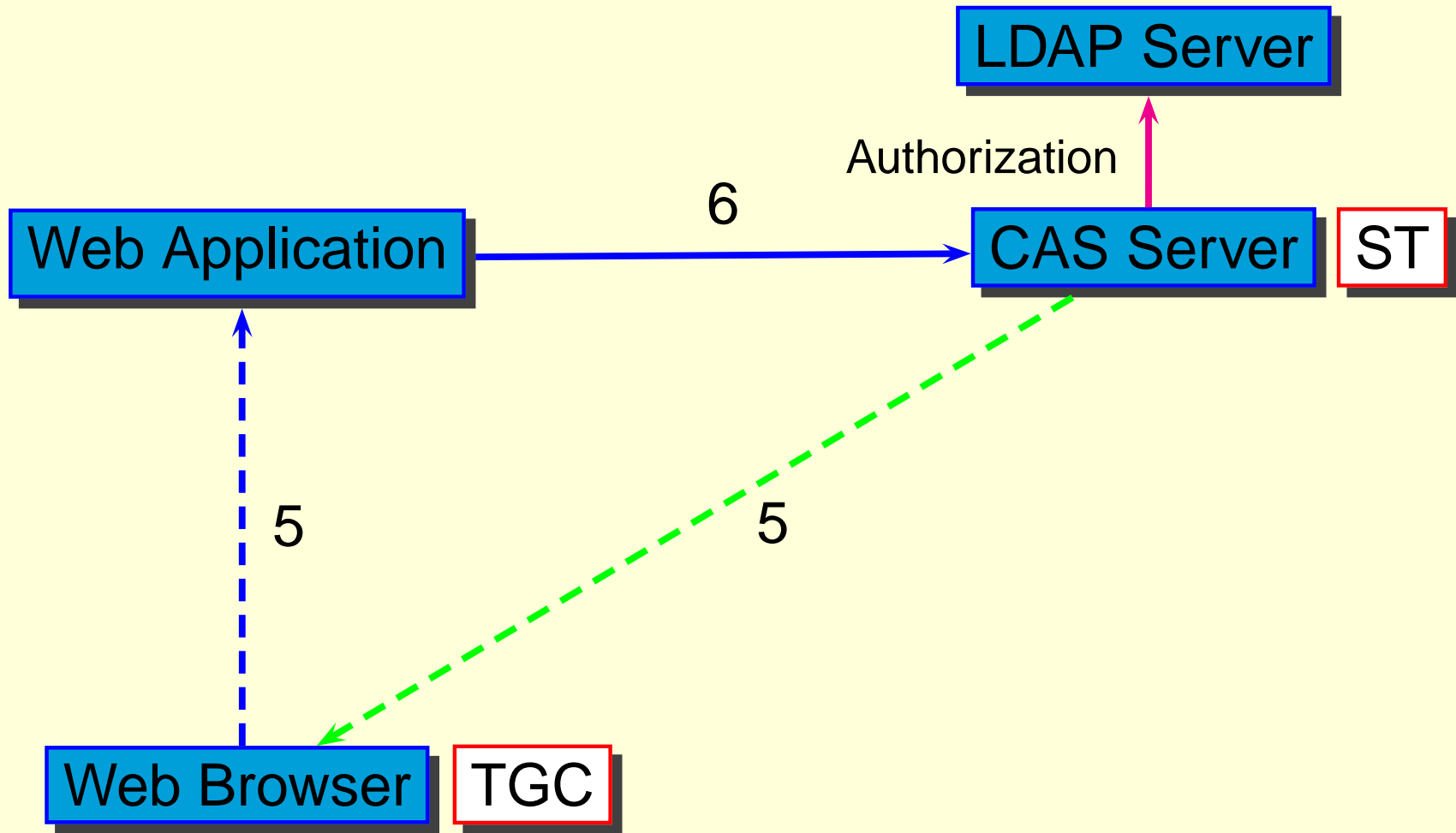
4. Send Ticket Granting Cookie to Browser

# CAS 認証のしくみ (1: Login (5))



5. Redirect to <https://aFQDN/a.html&ticket=ST-xxx>

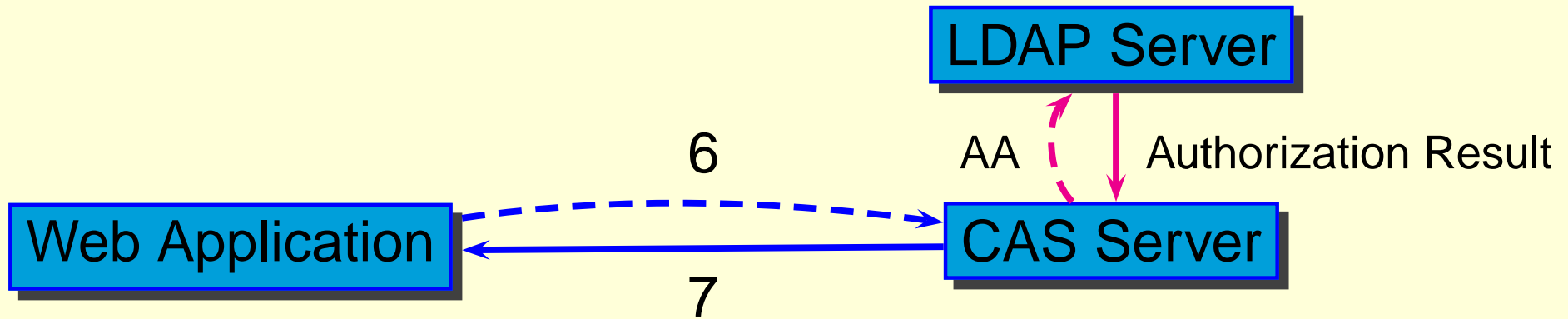
# CAS 認証のしくみ (1: Login (6))



6. Verify Service Ticket



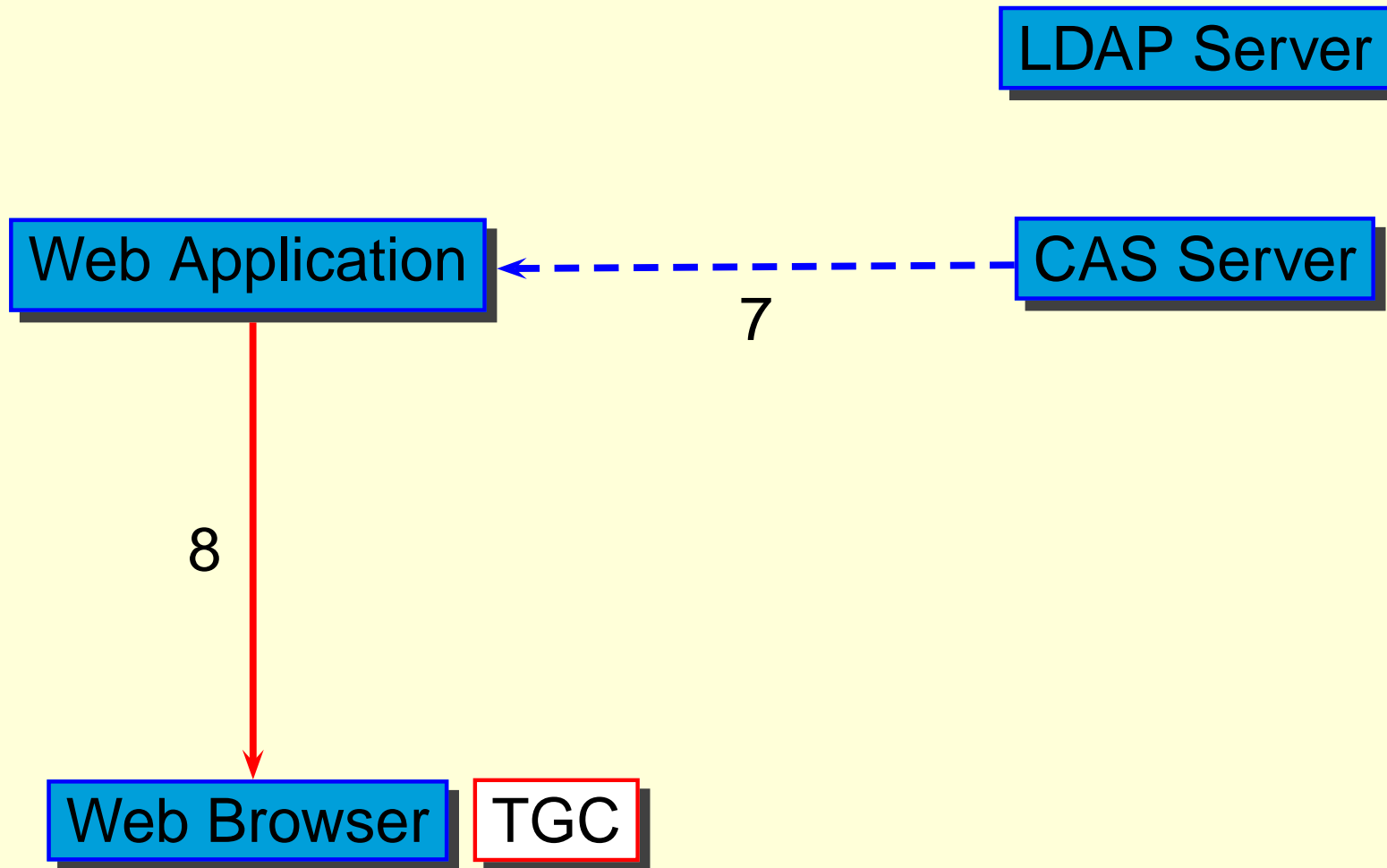
# CAS 認証のしくみ (1: Login (7))



**Web Browser** **TGC**

7. Receive verify result form CAS server

# CAS 認証のしくみ (1: Login (8))



8. Receive Data from Application Server

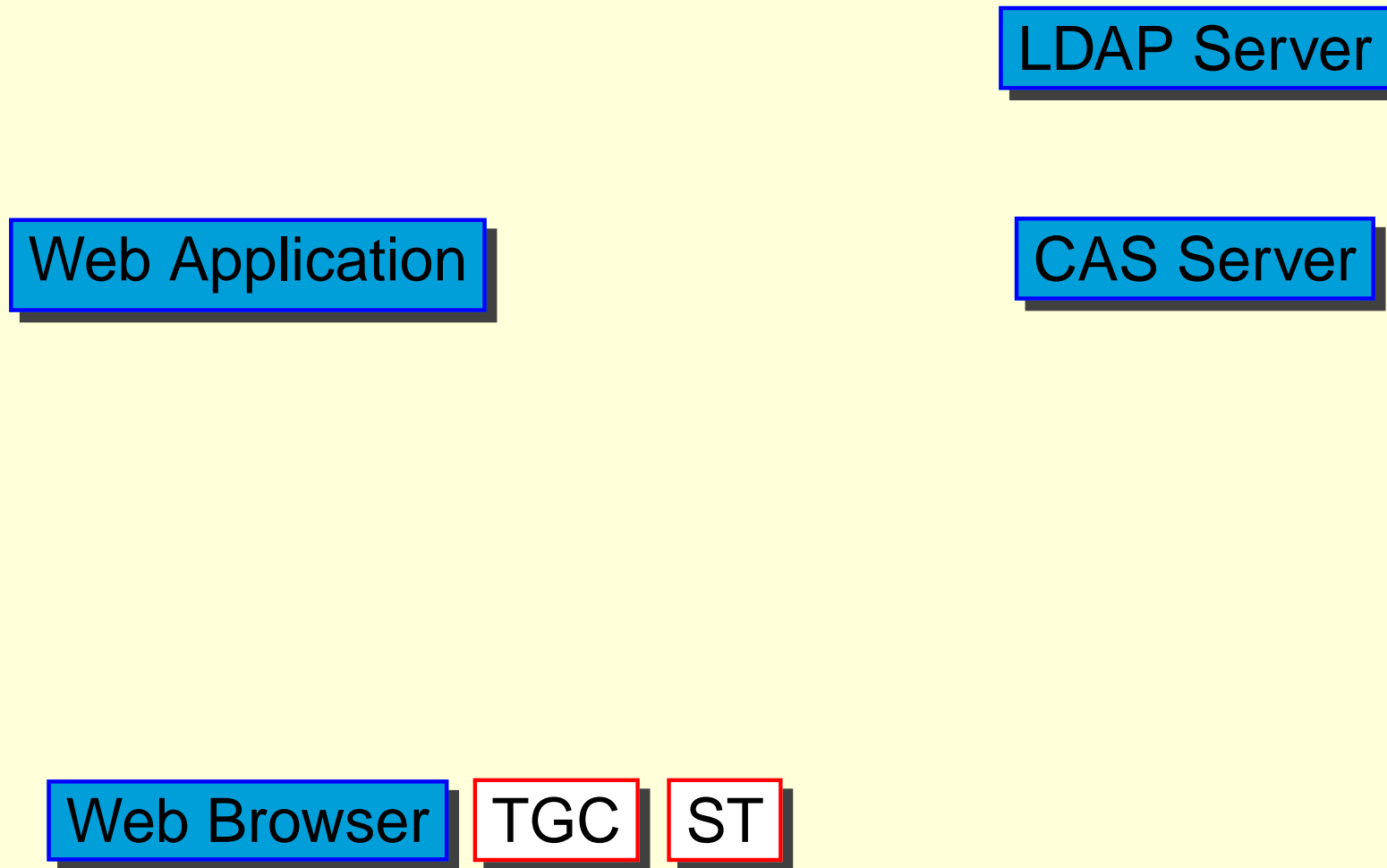
# CAS 認証のしくみ (注意事項)

- Login の一連の操作
  - 一見すると何度もアクセスが発生している
  - JavaScript/HTTP redirection が多数を占める
- 実際にユーザから visible なアクセス：以下の 2 回
  - Login Window
  - 実際のページの取得

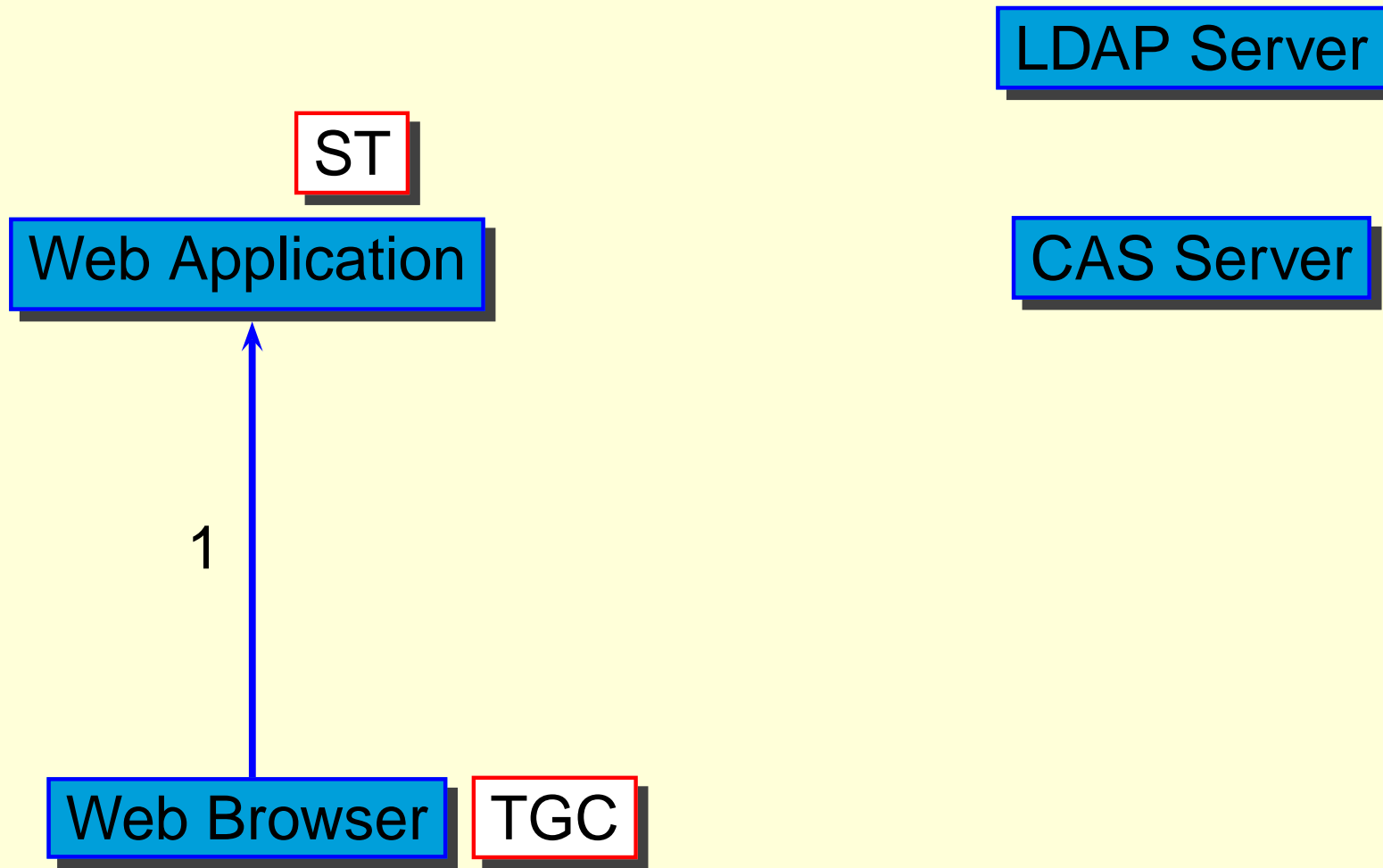
# CAS 認証のしくみ (2: Verify Ticket)

- Login 後のアクセス
  - ST による Authorization
  - 異なる “Service Class” へのアクセス時には TGC を検証
  - アクセスごとに TGC の “count down timer” を更新
- ST が Timeout している時
  - Login への redirection を発行
  - Authorization 後に新規 ST を発行
- ST による Authorization に失敗したとき
  - “CAS Message Page” への redirect を発行

# CAS 認証のしくみ (2: Verify Ticket (0))

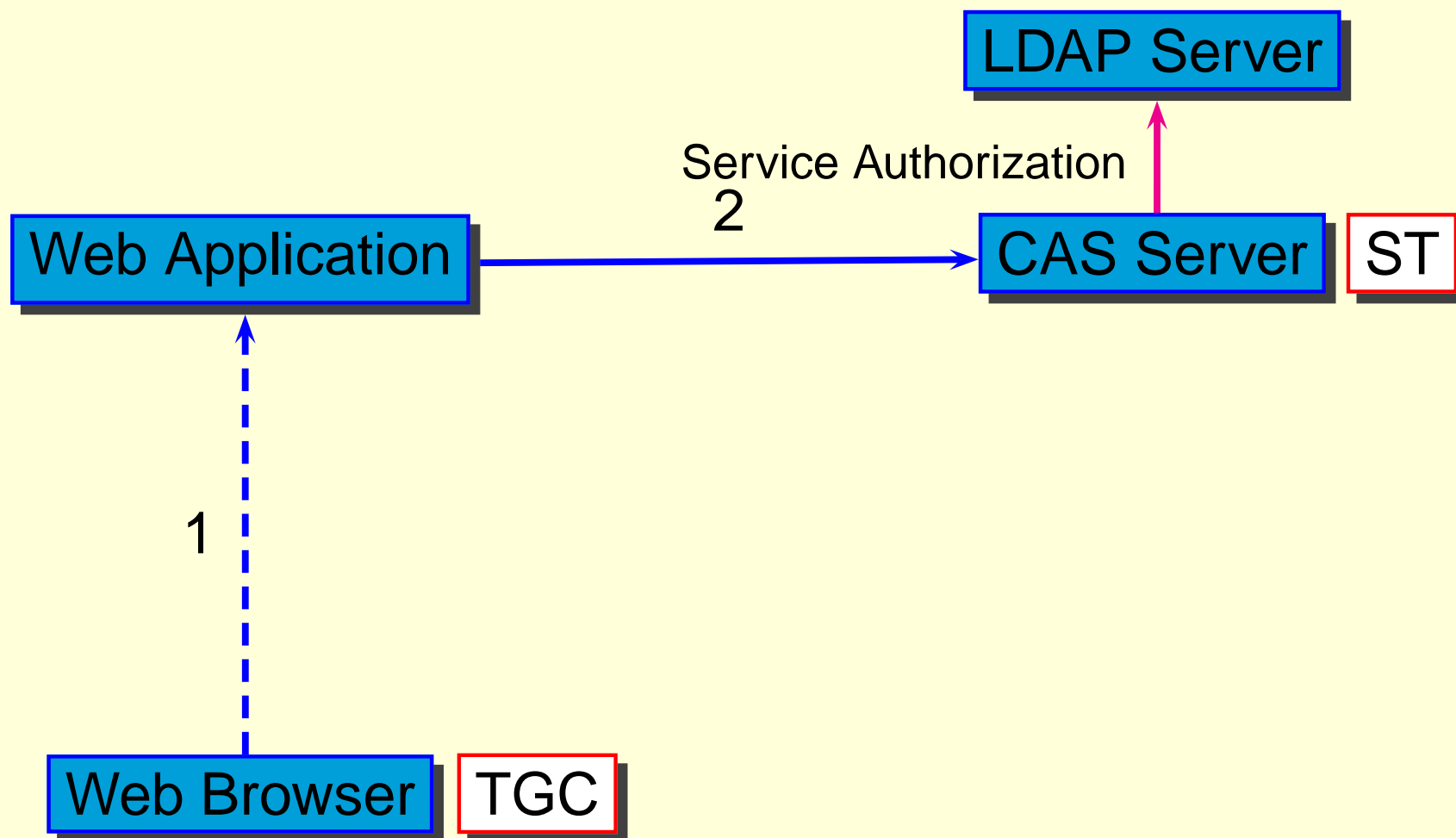


# CAS 認証のしくみ (2: Verify Ticket (1))



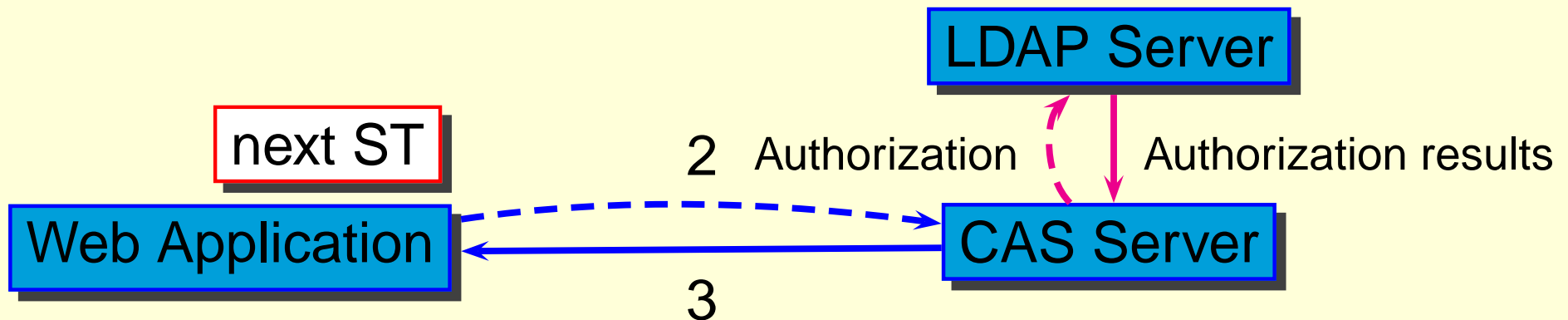
1. Access to <https://aFQDN/a.html&ticket=ST-xxxxx>

## CAS 認証のしくみ (2: Verify Ticket (2))



2. Verify `ticket=ST-xxxxx` with `service=https://aFQDN/a.html`

## CAS 認証のしくみ (2: Verify Ticket (3))

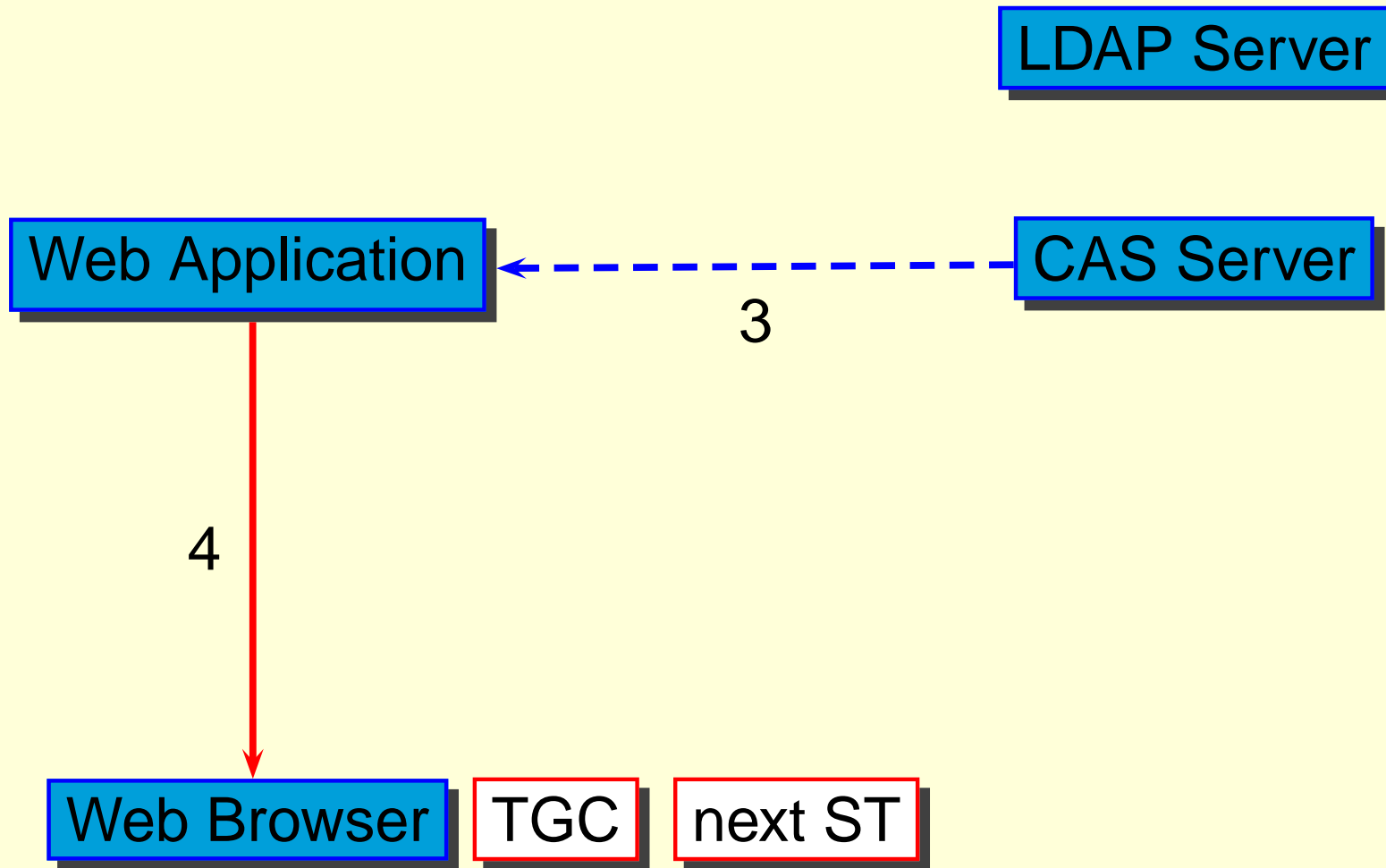


Web Browser TGC

3. Get authorization results, user information and **NEXT TICKET**



## CAS 認証のしくみ (2: Verify Ticket (4))

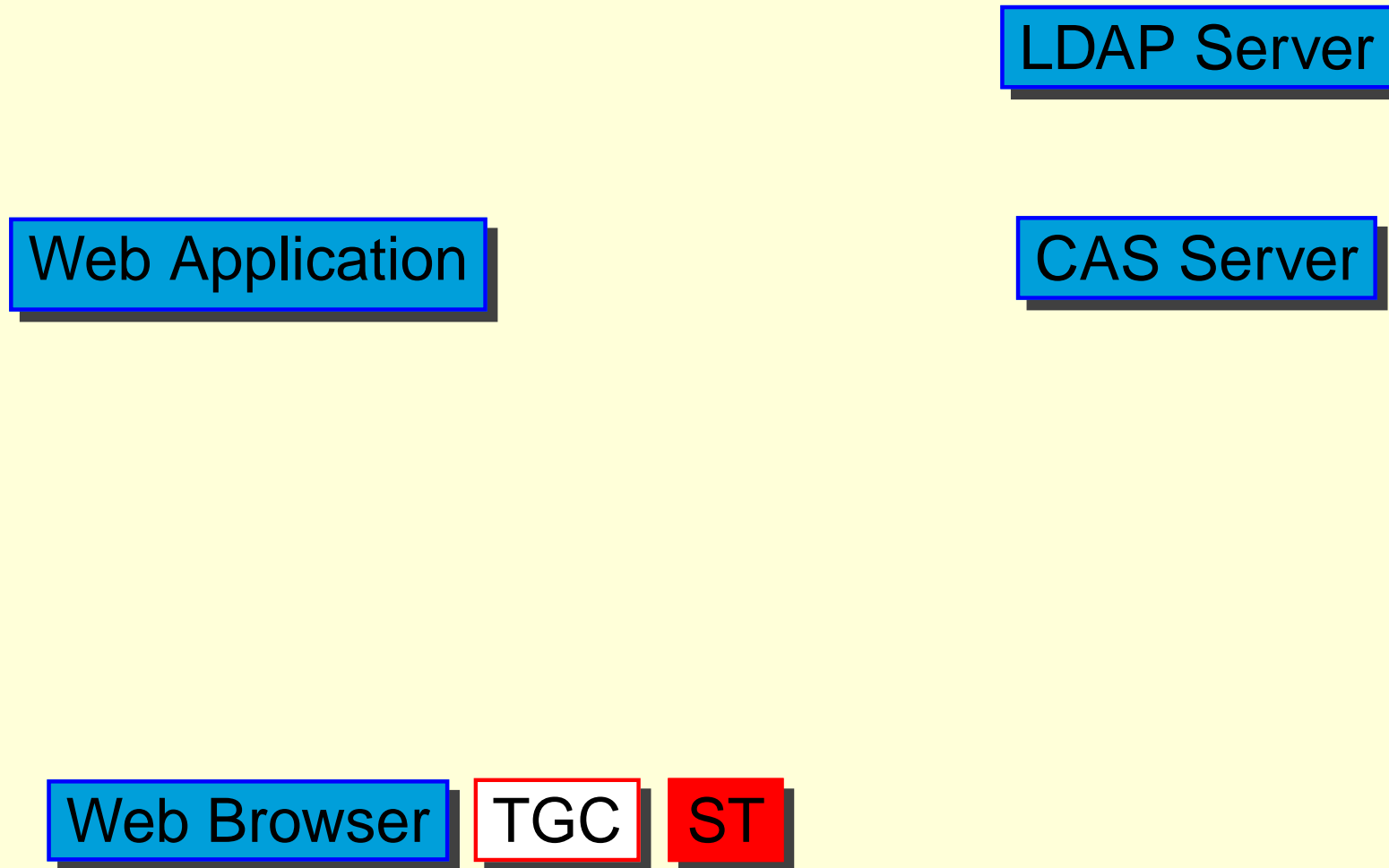


4. Reply from Web Application with **NEXT TICKET**

# CAS 認証のしくみ (3: Fail to Verify Ticket)

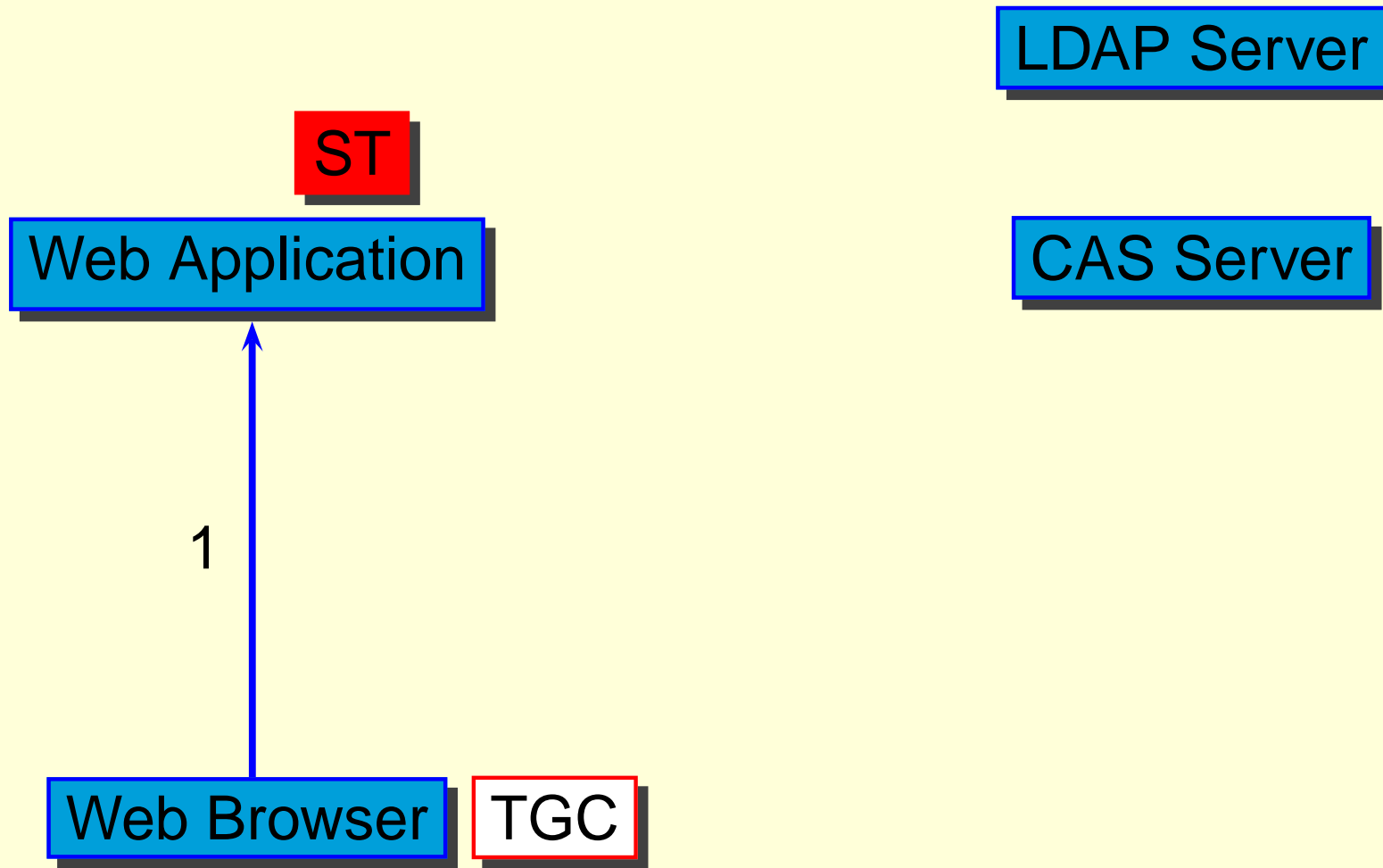
- Service Ticket が **INVALID TICKET** となる
  - Service Ticket が Timeout している
  - 異なる “Service Class” へのアクセス
- Ticket Granting Cookie を検証
- 新規 Service Ticket の発行

# CAS 認証のしくみ (3: Fail to Verify Ticket (0))



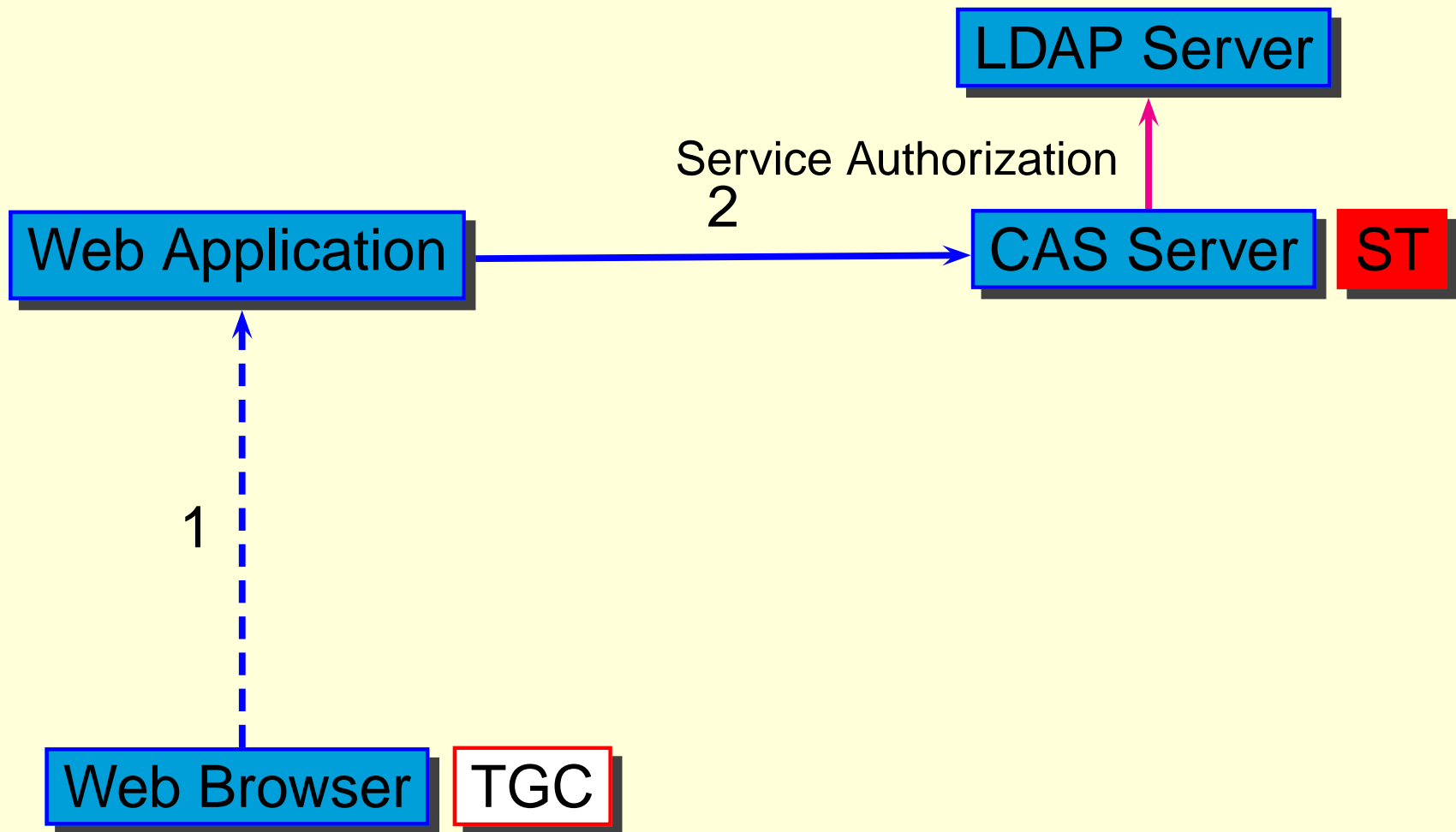
ST is expired or belonged to different ACCESS CLASS

# CAS 認証のしくみ (3: Fail to Verify Ticket (1))



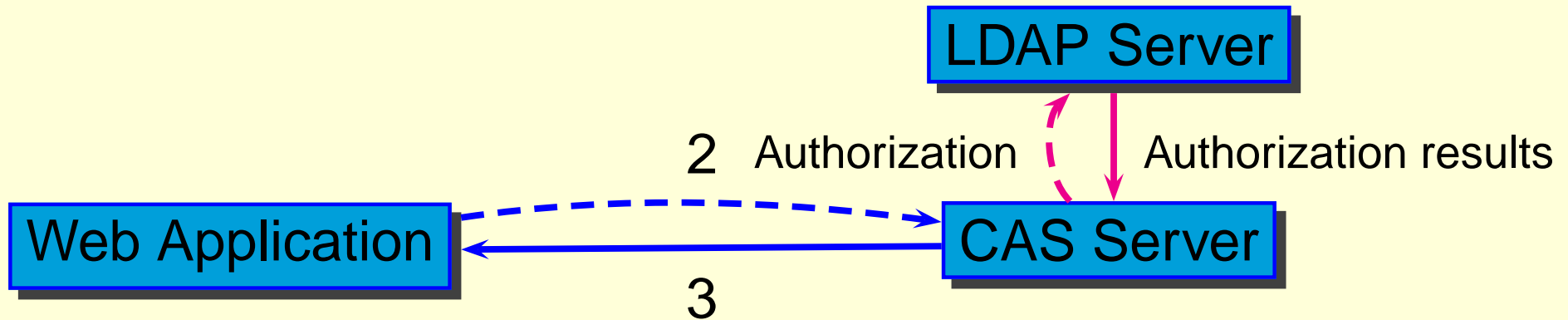
1. Access to <https://aFQDN/a.html&ticket=ST-xxxxx>

## CAS 認証のしくみ (3: Fail to Verify Ticket (2))



2. Verify `ticket=ST-xxxxxx` with `service=https://aFQDN/a.html`

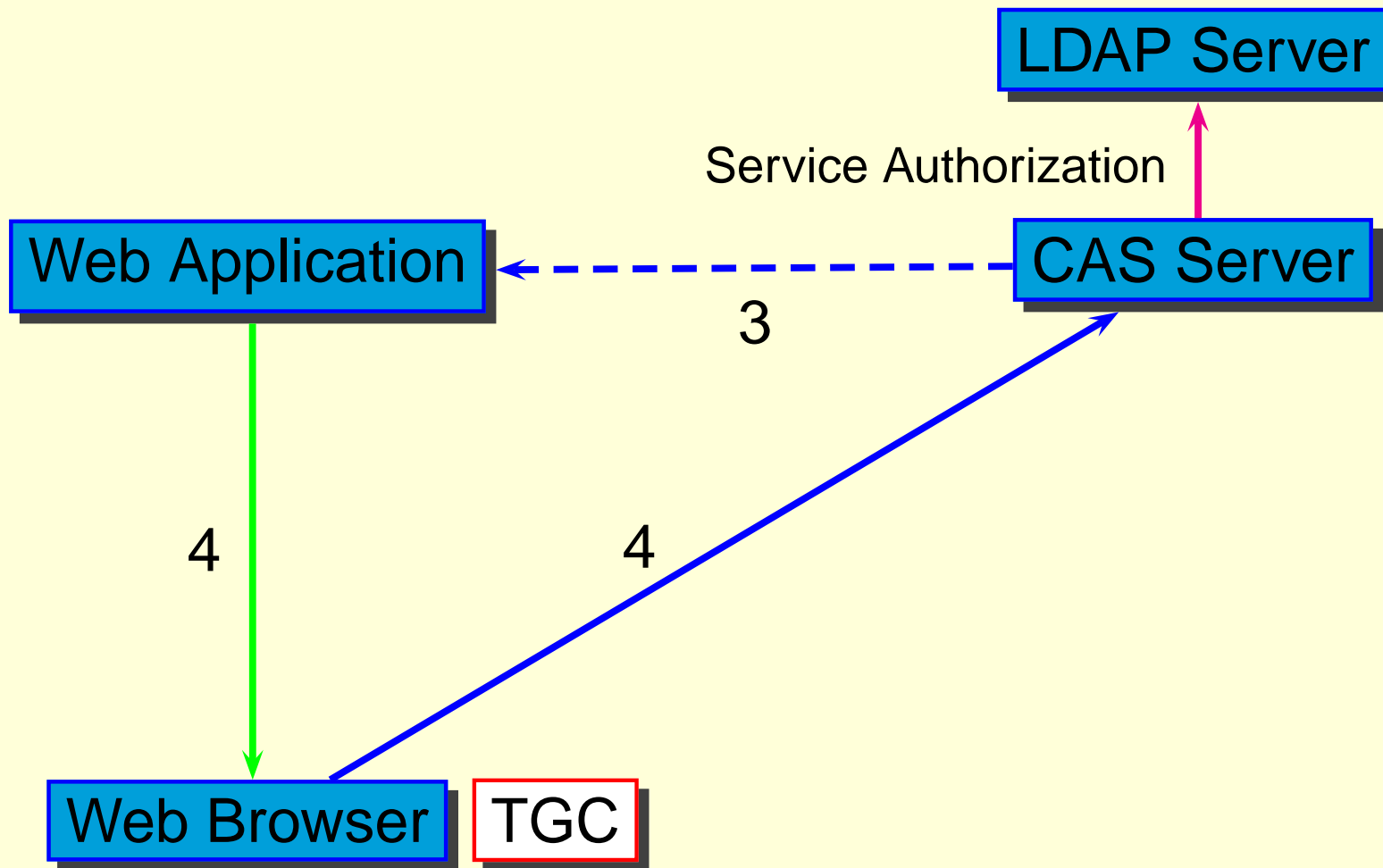
## CAS 認証のしくみ (3: Fail to Verify Ticket (3))



Web Browser TGC

3. Get authorization result: **INVALID TICKET**

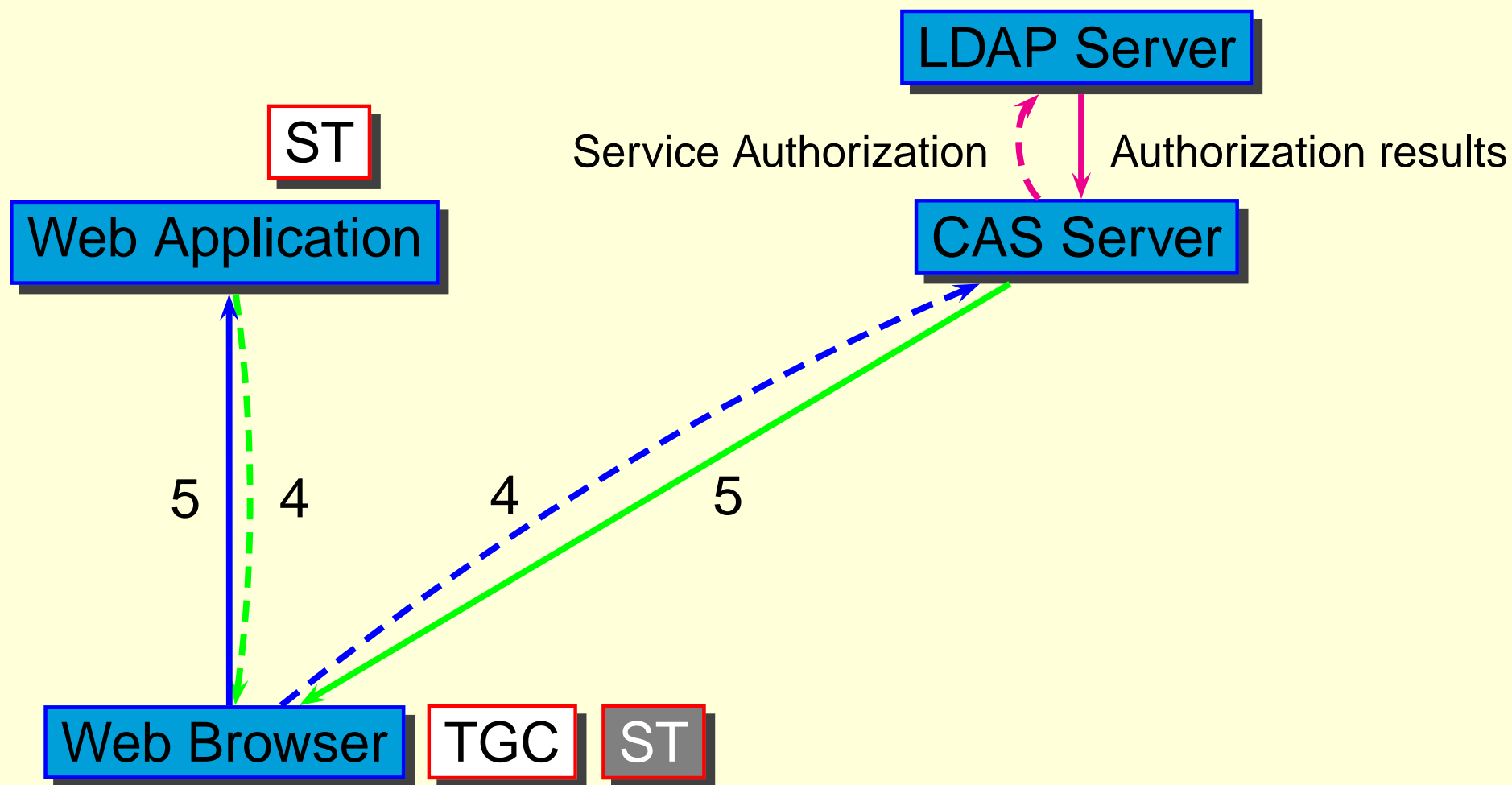
## CAS 認証のしくみ (3: Fail to Verify Ticket (4))



4. Redirect to

<https://CAS/login&service=https://aFQDN/a.html>

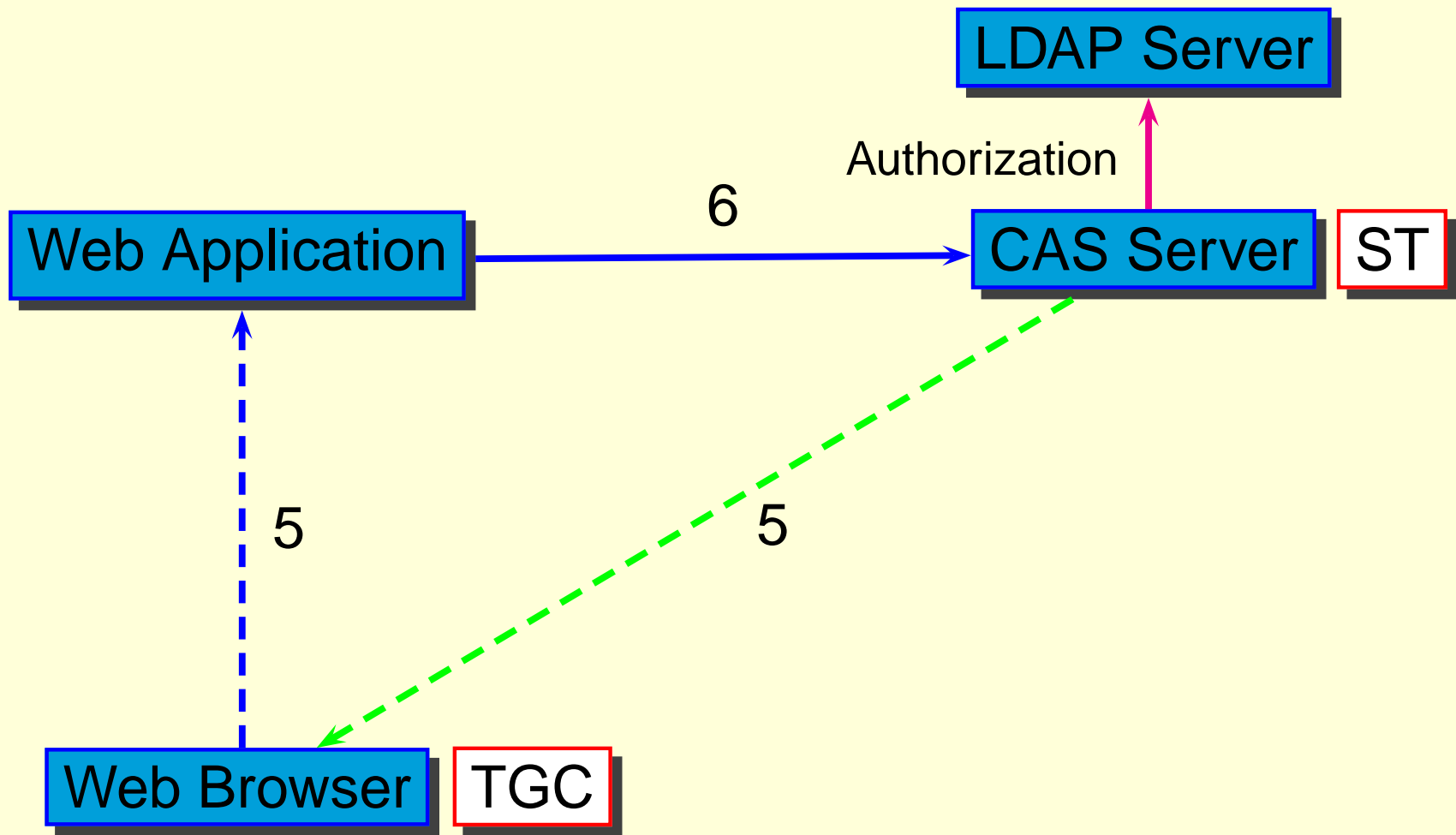
## CAS 認証のしくみ (3: Fail to Verify Ticket (5))



5. Redirect to <https://aFQDN/a.html&ticket=ST-xxx>

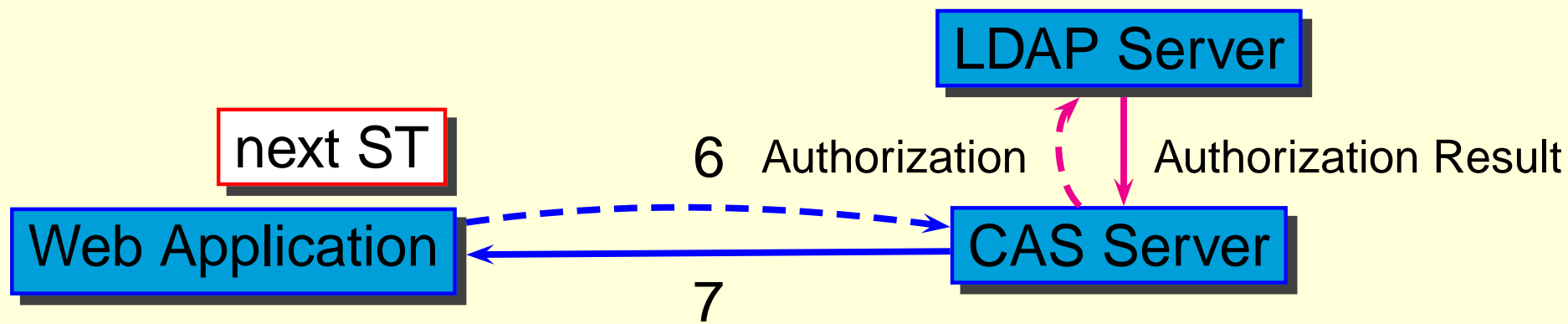


## CAS 認証のしくみ (3: Fail to Verify Ticket (6))



6. Verify Service Ticket

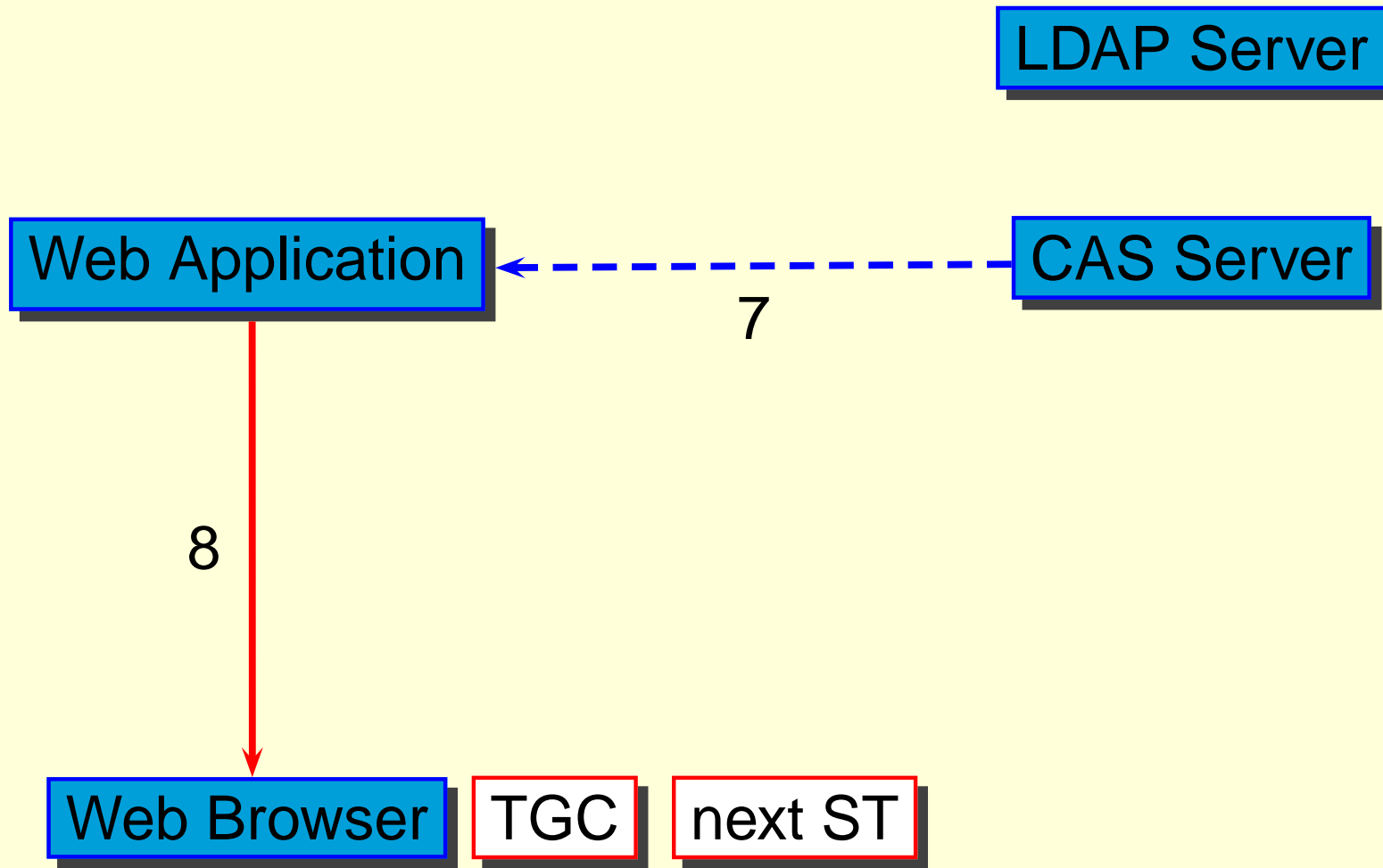
## CAS 認証のしくみ (3: Fail to Verify Ticket (7))



Web Browser TGC

7. Receive verify result form CAS server

## CAS 認証のしくみ (3: Fail to Verify Ticket (8))

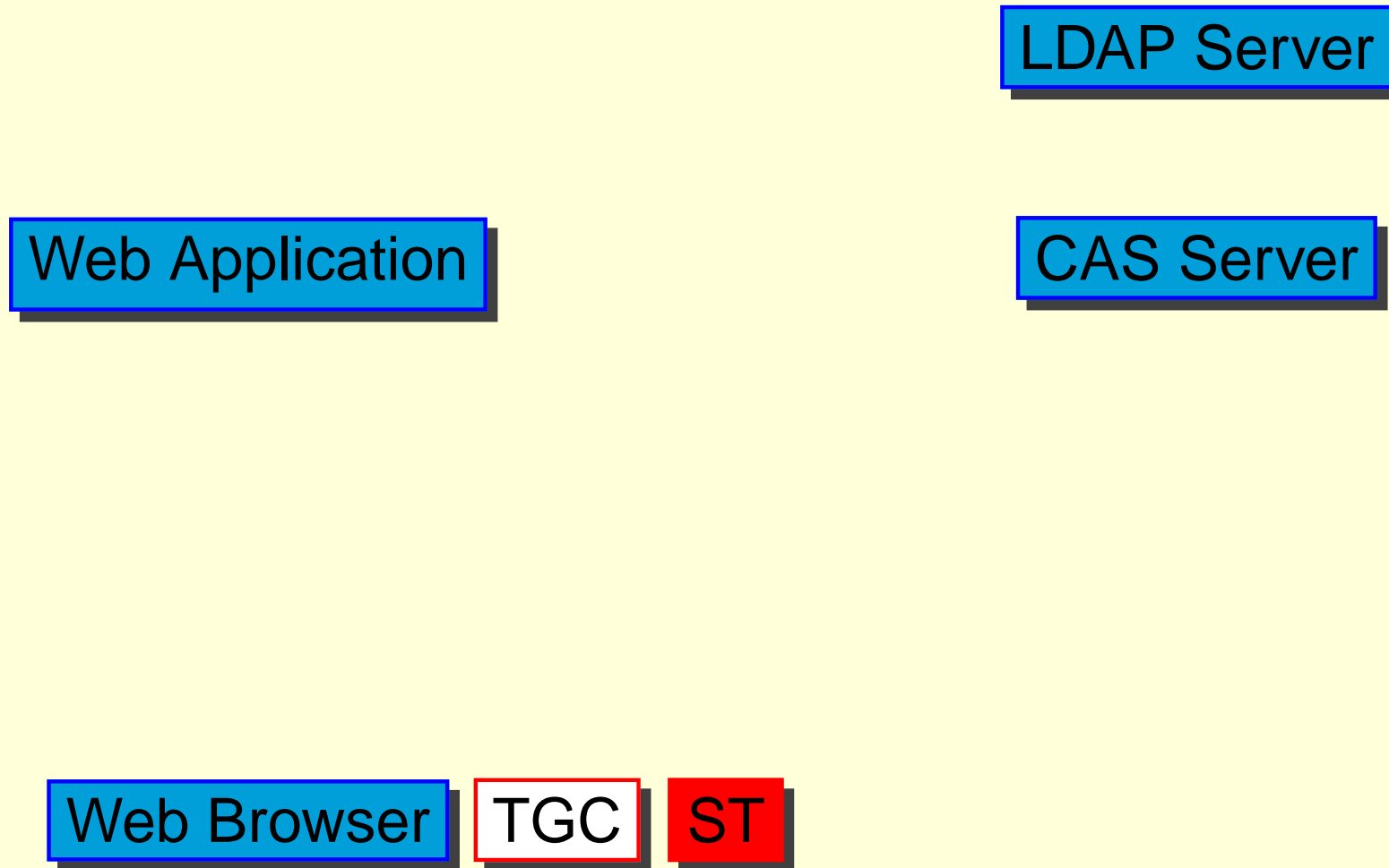


8. Receive Data from Application Server

# CAS 認証のしくみ (4: Fail to Authorization)

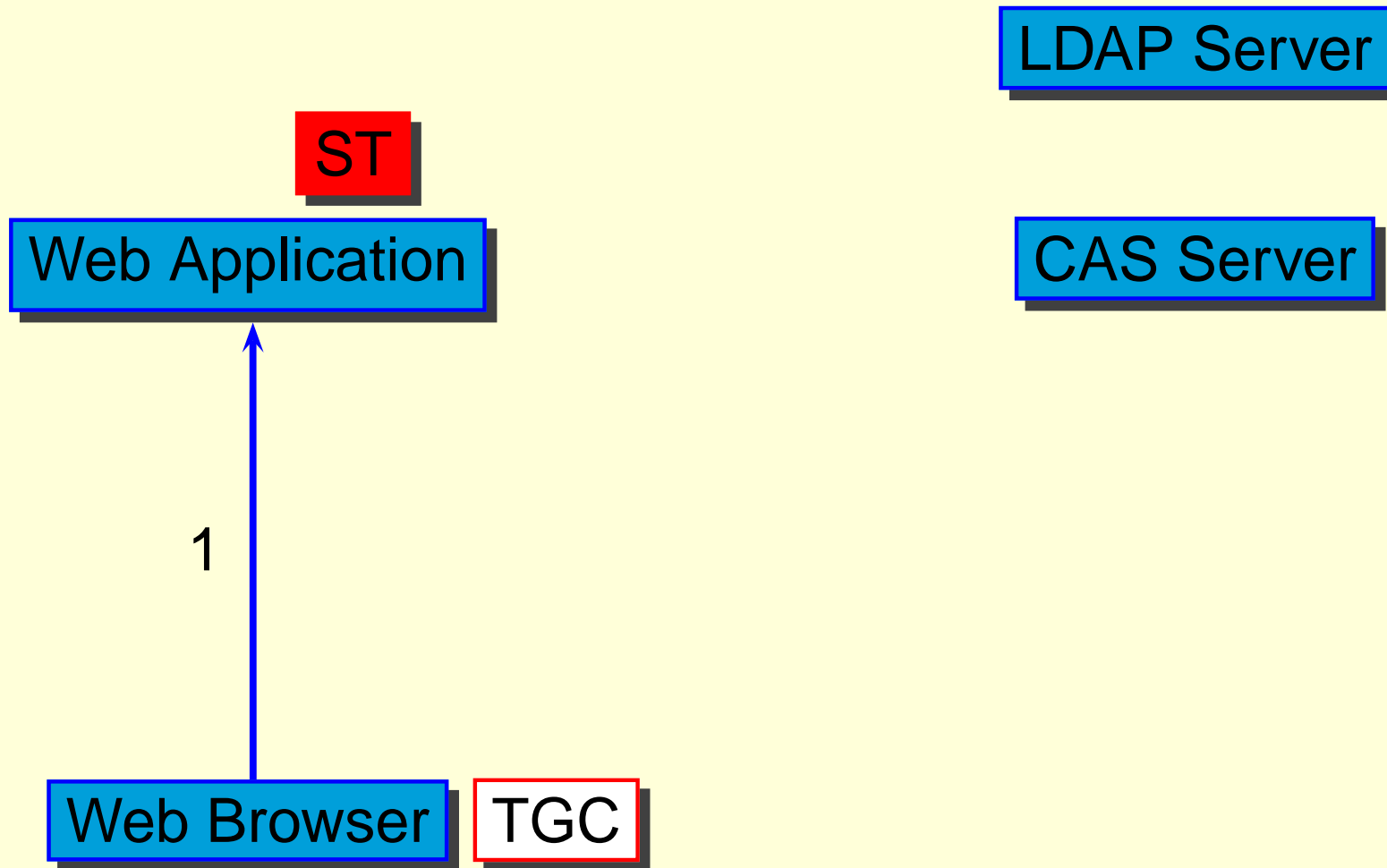
- Service Ticket が **ACCESS DENIED** となる
  - ST に記載の Service (URL) へのアクセス権がない
  - ST に記載されたデータ ∈ CAS Server 内のデータベース
- 「CAS Message Page」を表示

# CAS 認証のしくみ (4: Fail to Authorization (0))



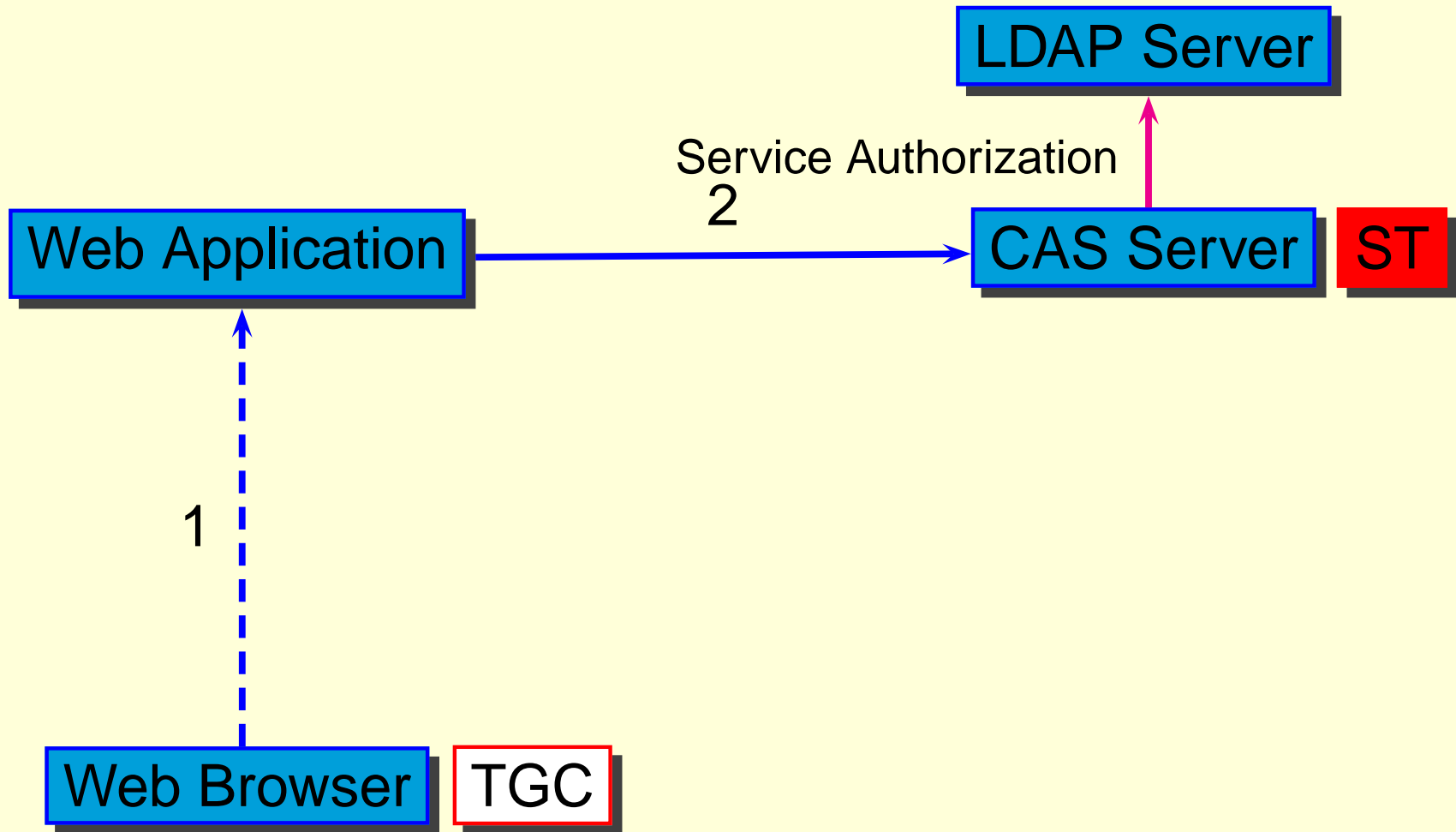
If SERVICE is **denied to access**

# CAS 認証のしくみ (4: Fail to Authorization (1))



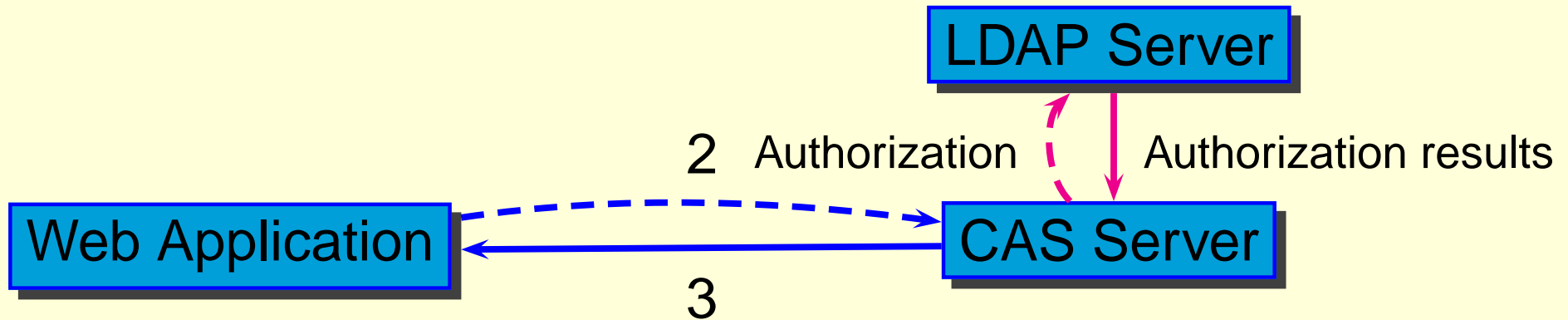
1. Access to <https://aFQDN/a.html&ticket=ST-xxxxx>  
If SERVICE is **denied to access**

## CAS 認証のしくみ (4: Fail to Authorization (2))



2. Verify `ticket=ST-xxxxxx` with `service=https://aFQDN/a.html`

## CAS 認証のしくみ (4: Fail to Authorization (3))

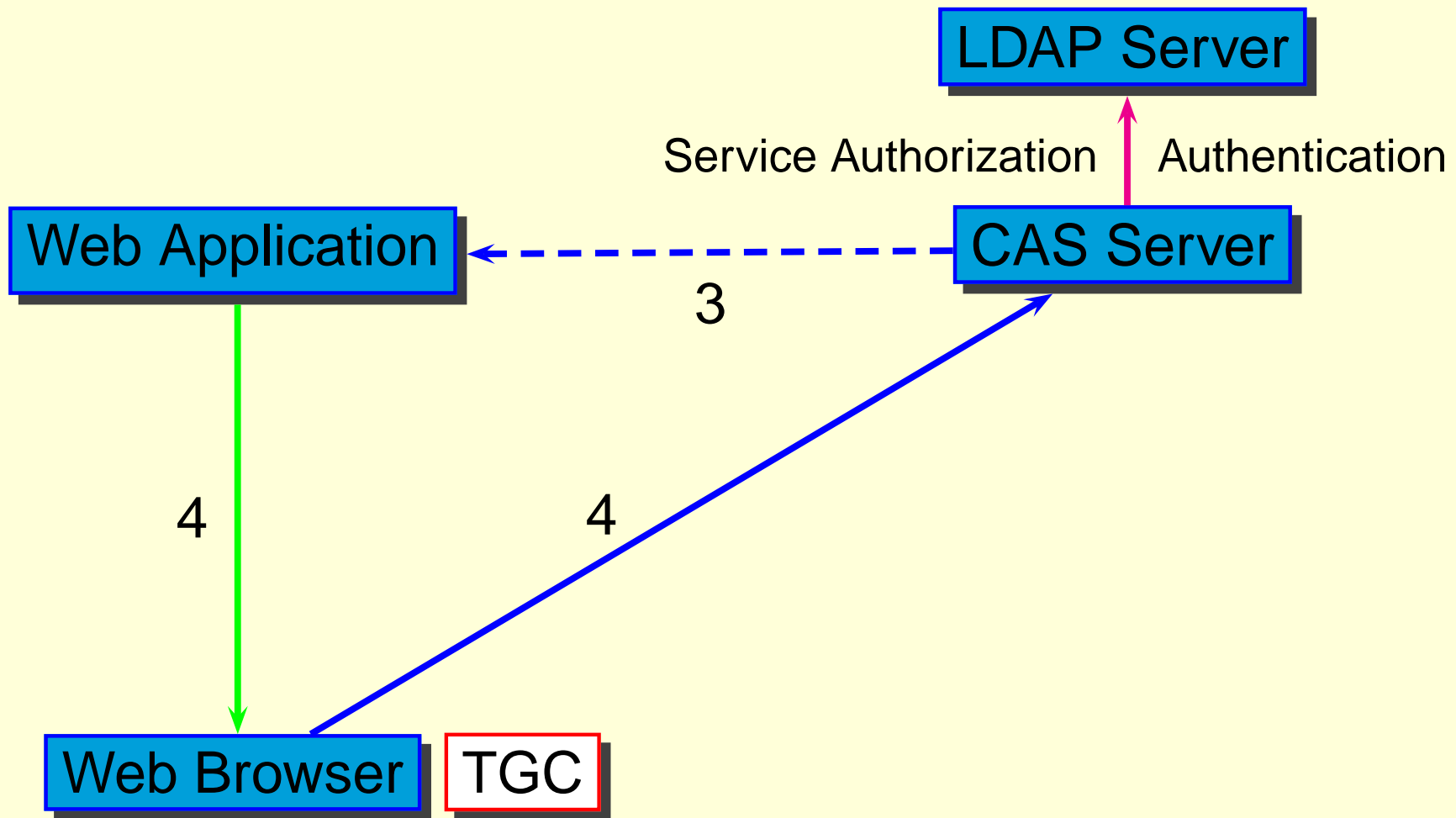


Web Browser TGC

3. Get authorization result: **INVALID TICKET**



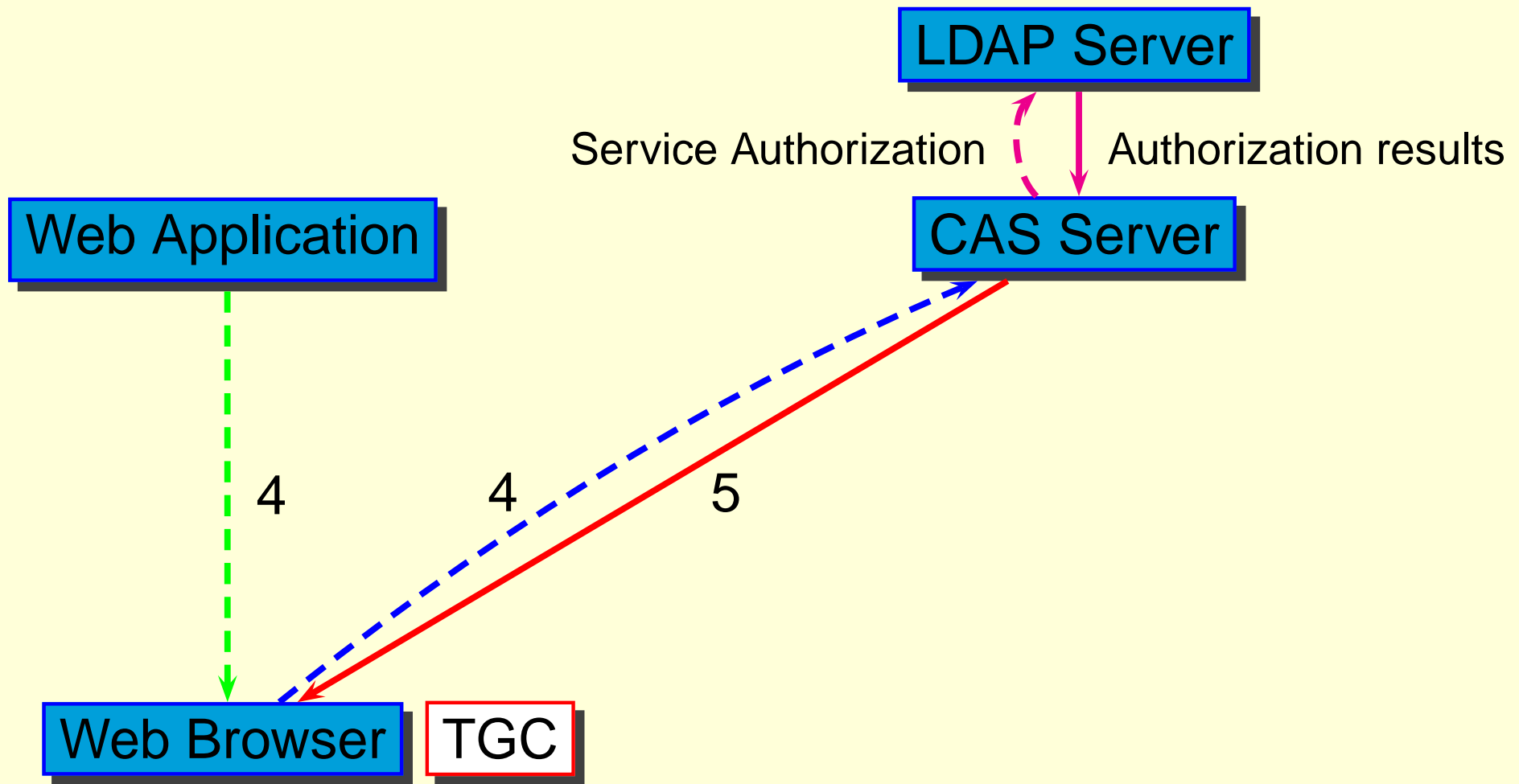
# CAS 認証のしくみ (4: Fail to Authorization (4))



4. Redirect to

<https://CAS/login&service=https://aFQDN/a.html>

# CAS 認証のしくみ (4: Fail to Authorization (5))



5. ACCESS DENIED

## CAS 認証のしくみ (5: If TGC is expired)

- Ticket Granting Cookie (TGC) が expired した状態で「サービス」にアクセス
  - Login ページへ redirect
  - 認証後「サービス」にアクセス可能

# CAS 認証のしくみ (5: If TGC is expired (0))

LDAP Server

Web Application

CAS Server

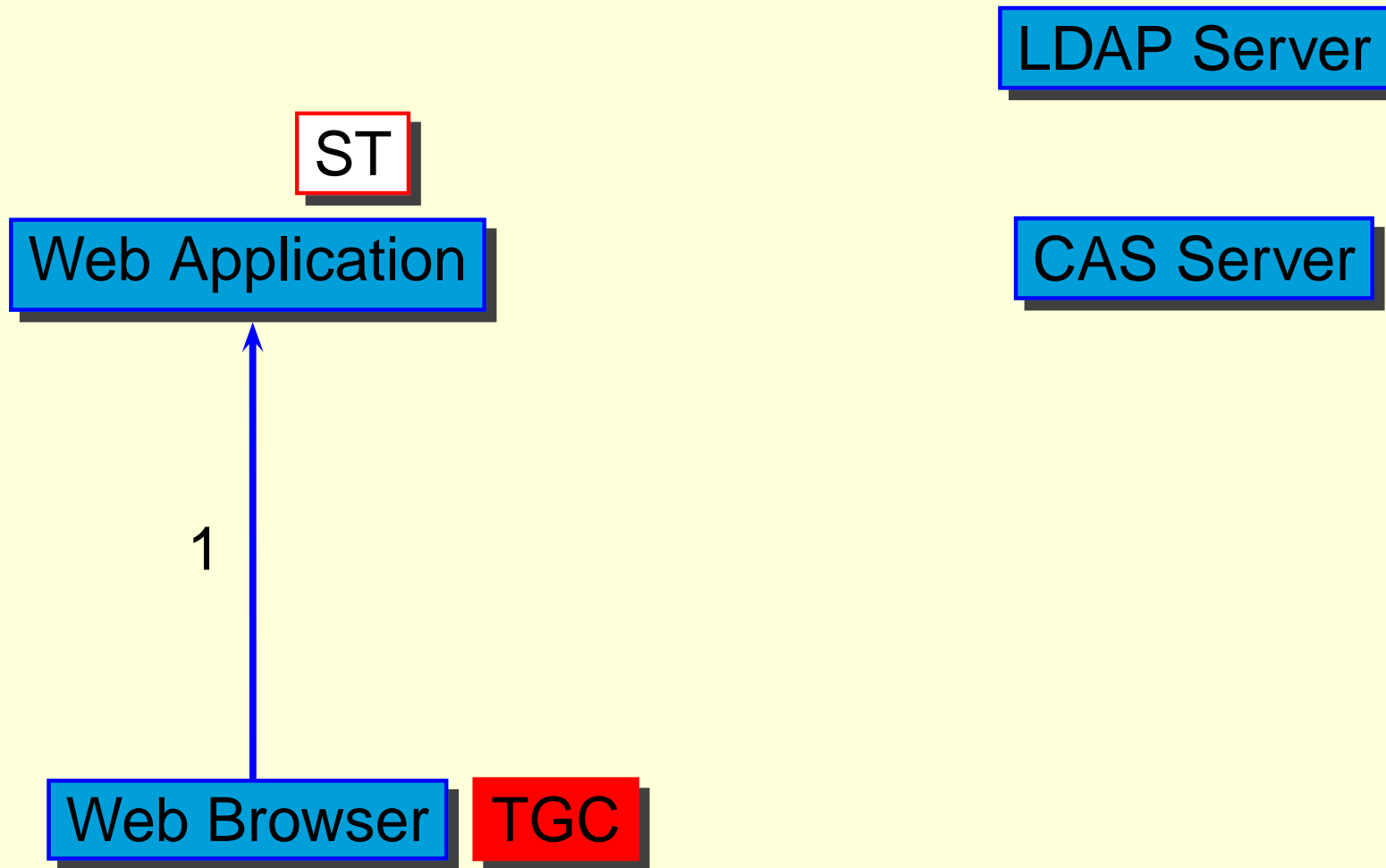
Web Browser

TGC

ST

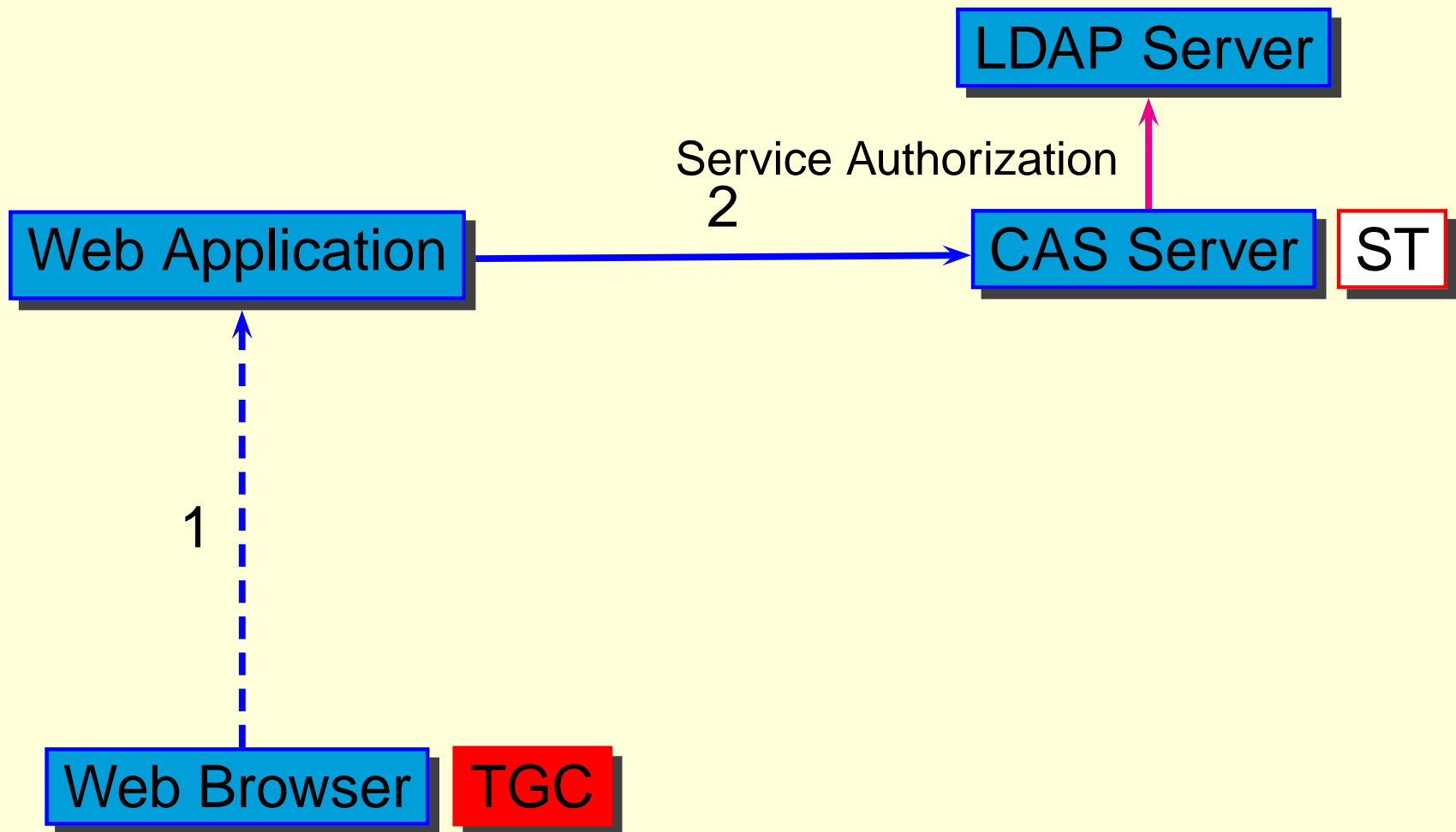
If TGC is expired

# CAS 認証のしくみ (5: If TGC is expired (1))



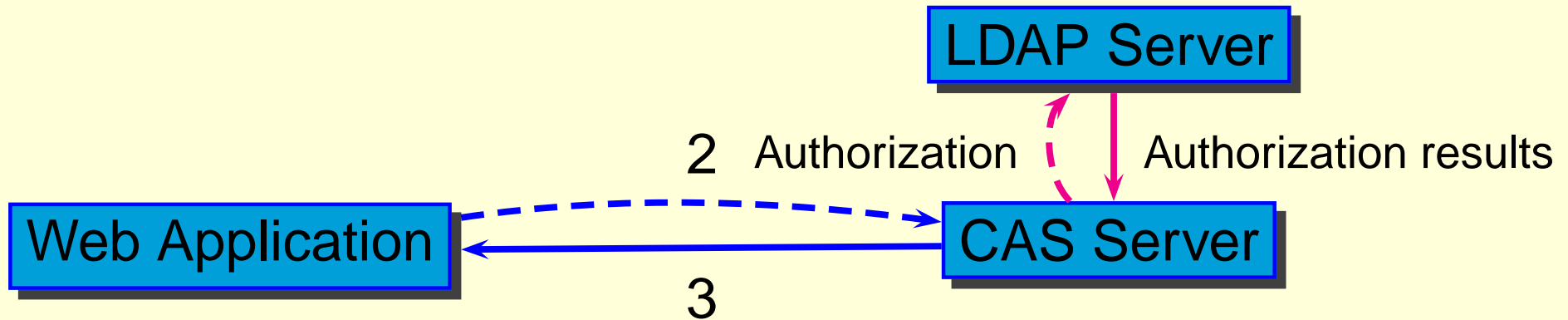
1. Access to <https://aFQDN/a.html&ticket=ST-xxxxxx>

## CAS 認証のしくみ (5: If TGC is expired (2))



2. Verify `ticket=ST-xxxxx` with `service=https://aFQDN/a.html`

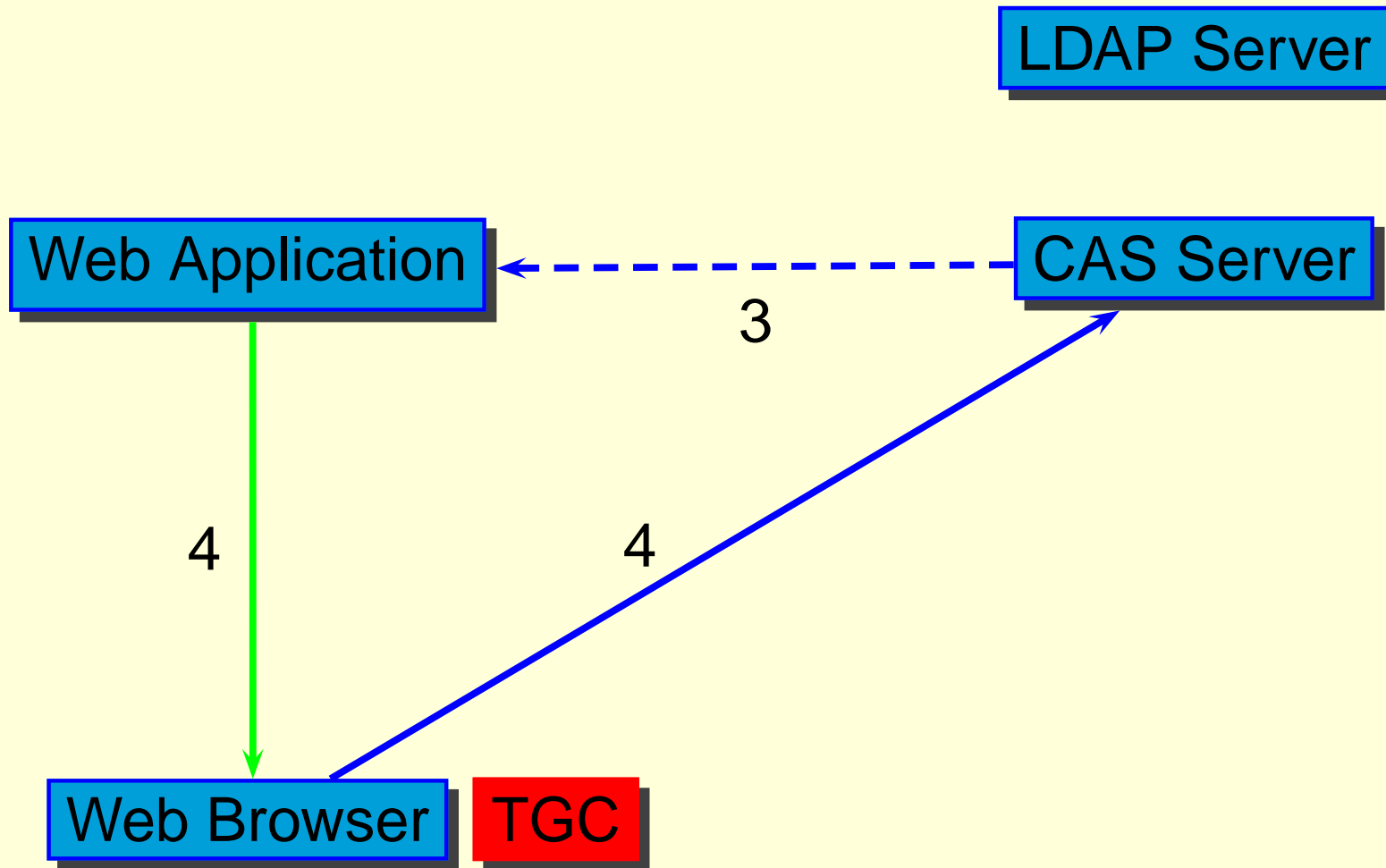
## CAS 認証のしくみ (5: If TGC is expired (3))



Web Browser TGC

3. Get authorization result: **INVALID TICKET**  
since parent TGC is expired

## CAS 認証のしくみ (5: If TGC is expired (4))

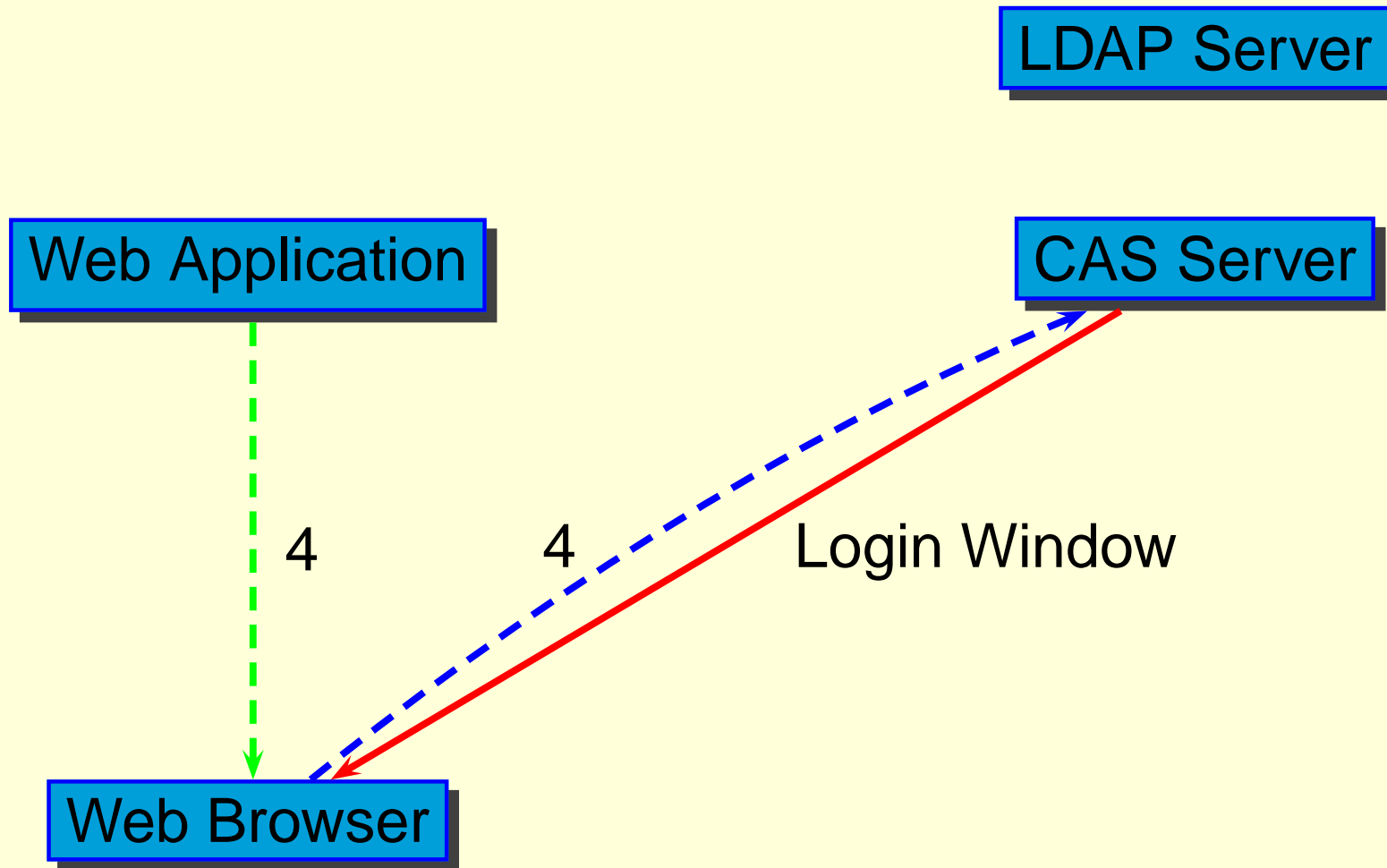


4. Redirect to

<https://CAS/login&service=https://aFQDN/a.html>



## CAS 認証のしくみ (5: If TGC is expired (5))



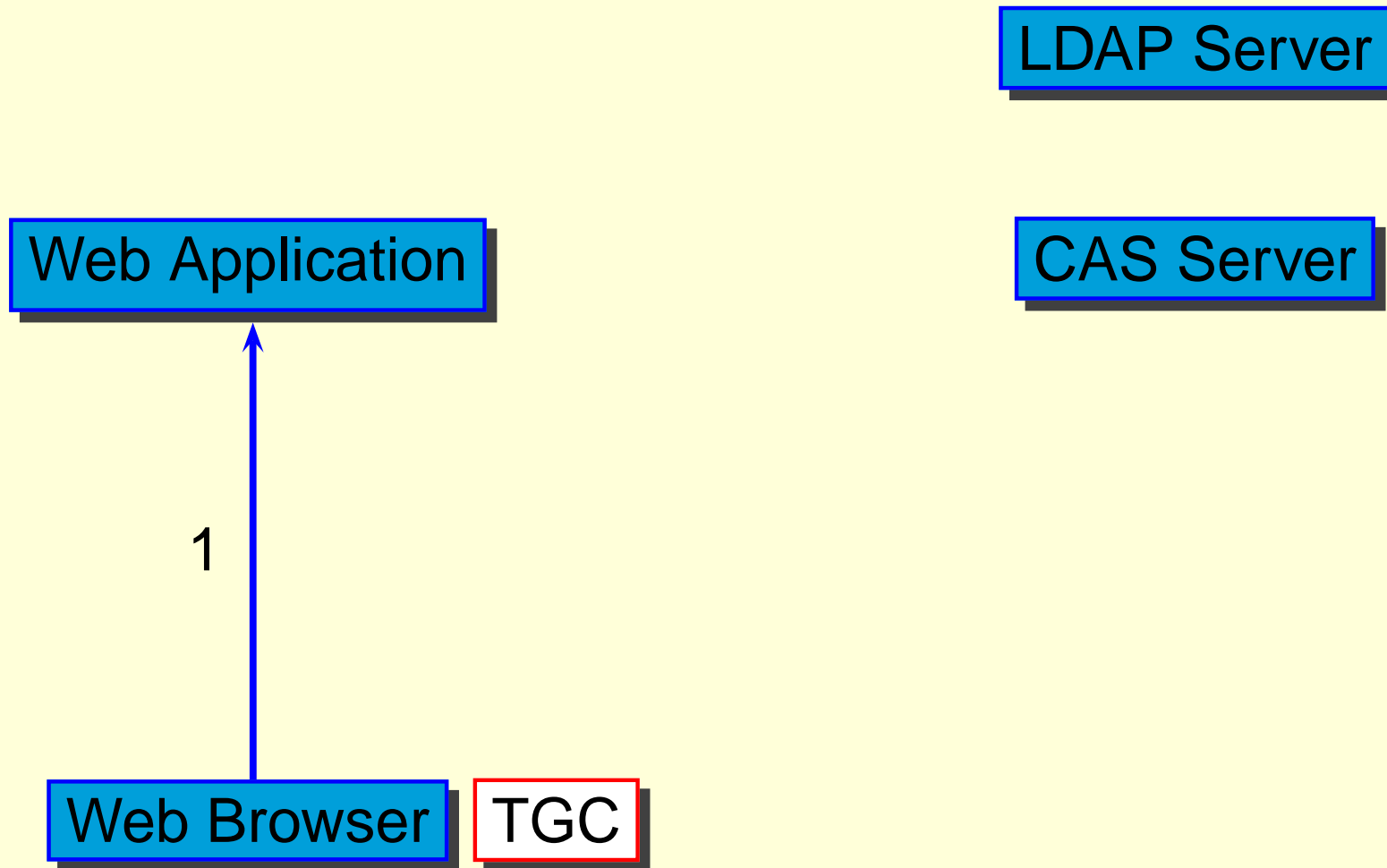
5. Redirect to <https://aFQDN/a.html&ticket=ST-xxx>

# CAS 認証のしくみ (6: Logout)

---

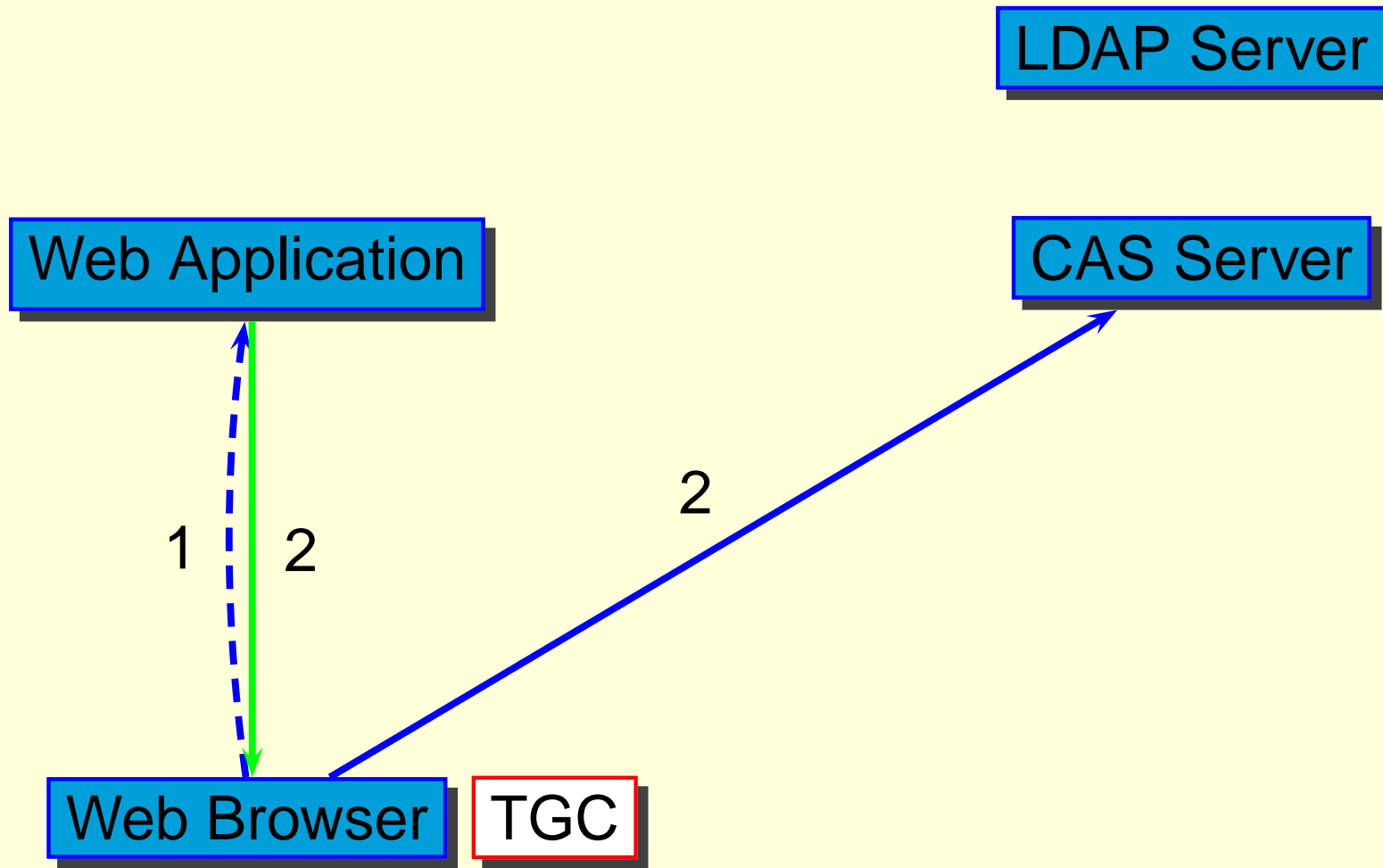
- Logout ボタン
  - Logout ページへのリンク
  - TGC を消去

# CAS 認証のしくみ (6: Logout (1))



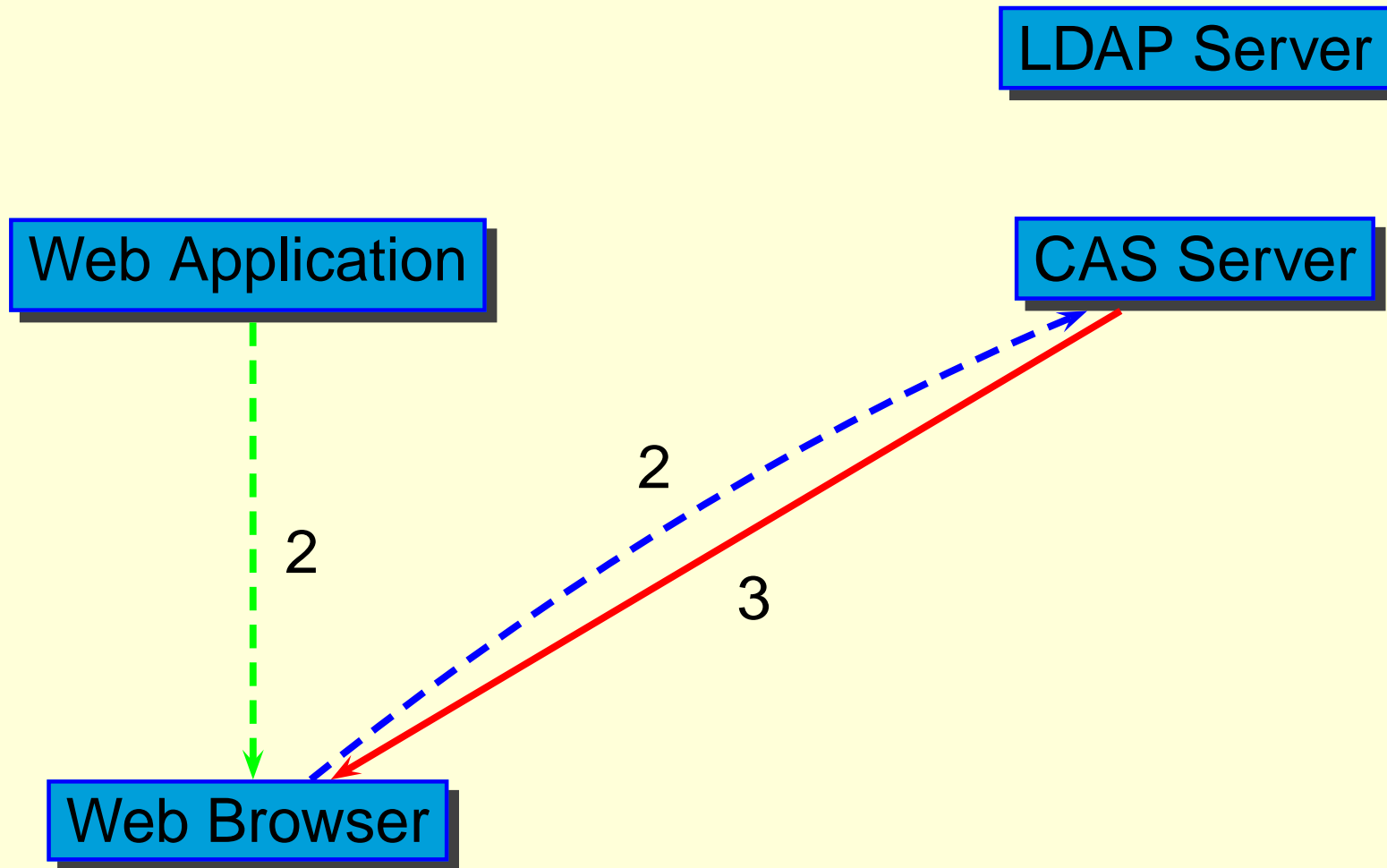
1. Access to <https://aFQDN/logout.html>

## CAS 認証のしくみ (6: Logout (2))



2. Redirect to <https://CAS/logout>

# CAS 認証のしくみ (6: Logout (3))



3. Delete TGC and Logout Message

# まとめ

- 従来 Authentication のみをサポートしていた  
Yale CAS Server に Authorization 機能を追加した
- 従来は GET method のみに対応したシステムだったが、  
POST method によるパラメータの受け渡しに成功
- XSS による TGC, ST などの流出に対応
- 更なる特徴：軽くて高速に動作
  - Yale-CAS: JAVA 2000 行弱, nu-CAS: JAVA 5000 行弱
  - 最大アクセス実績：4000 回/分
    - Sun Fire V480 (1.0GHz UltraSPAC III Cu x 2)
    - 4.0GB Memory
    - Solaris 8

# 将来にむけて

---

- 電子証明書によるクライアント認証の実装
- 完全な国際化対応
- WebDAV システムへの対応 (PUT method など)
- Open Source Software として公開