# IPv6 Ephemeral Addresses

<draft-kitamura-ipv6-ephemeral-address-00.txt>

# Harmless IPv6 Address State Extension (**Uncertain** State)

<draft-kitamura-ipv6-uncertain-address-state-00.txt>

Hiroshi KITAMURA

NEC Corporation

kitamura@da.jp.nec.com

# Prologue

- We propose two new ideas:
  - **Ephemeral Addresses**
  - **Uncertain Address State**

- They are ***small modification*** to the current specs.
- They are ***harmless*** and
  can ***coexist*** with current implementations.

But
We hope they bring much benefits to us.

# IPv6 Ephemeral Addresses

## <draft-kitamura-ipv6-ephemeral-address-00.txt>

Hiroshi KITAMURA
NEC Corporation
kitamura@da.jp.nec.com

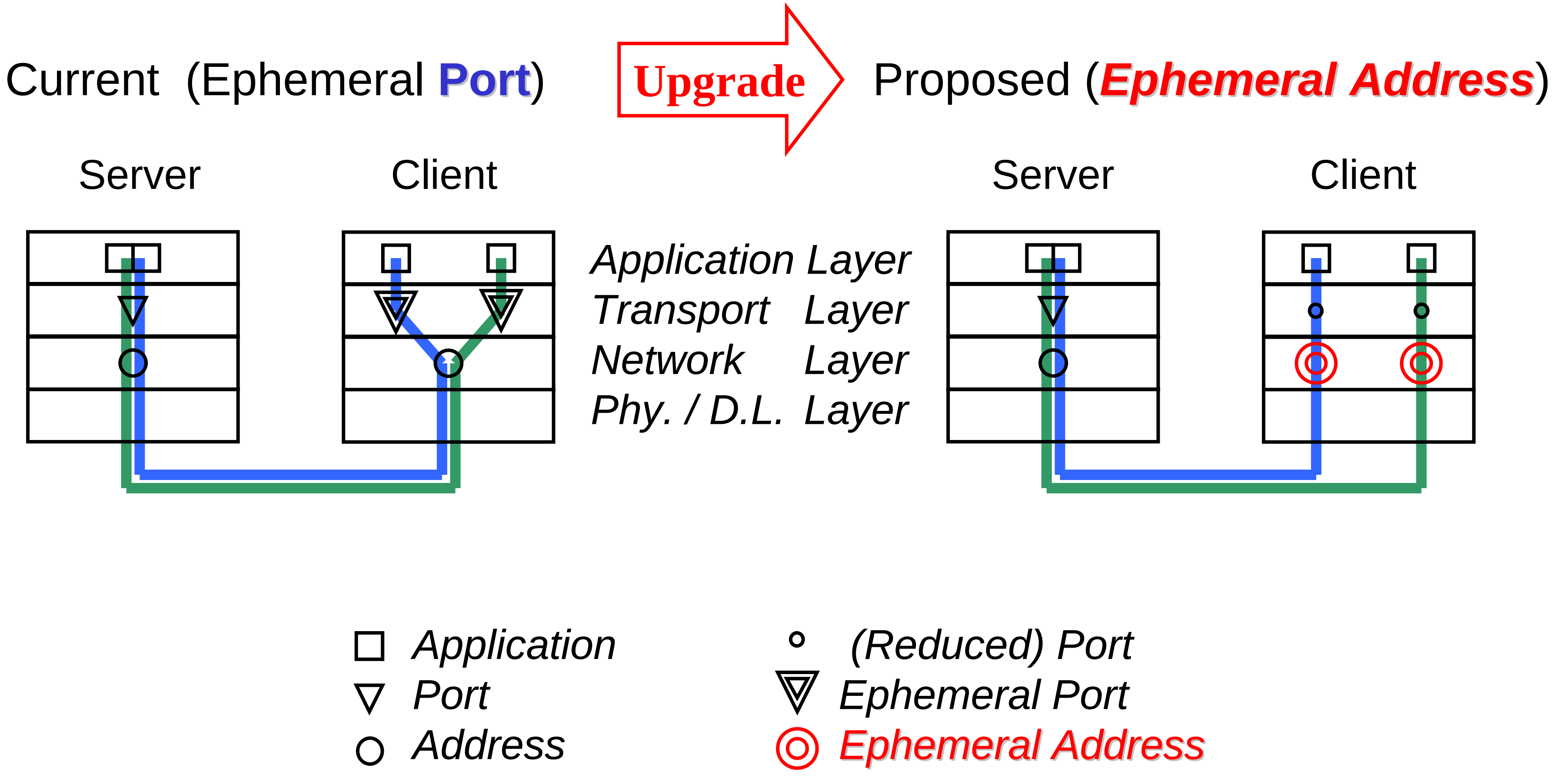
# Introduction of IPv6 Ephemeral Addresses

- “***Ephemeral Addresses***” are designed to be used as ***clients' source addresses*** of TCP / UDP sessions.

- “***Ephemeral Addresses***” are achieved by ***deriving*** from the existing “**Ephemeral Ports**” specifications.

- In other words:
  “***Ephemeral Addresses***” are achieved by
  ***naturally upgrading*** “**Ephemeral Ports**” concept
  from the **port** space to the **address** space.

# Basic Design of Ephemeral Addresses



# How Ephemeral Addresses Work

"**Ephemeral Addresses**" can contribute to various types of security enhancements (e.g., privacy protections etc.)

Definitions of "**Ephemeral Addresses**" are almost same as definitions of "Ephemeral Ports".

| | Ephemeral Ports | **Ephemeral Addresses** |
|---|---|---|
| Where used? | clients' source **ports** on the **transport layer** | clients' source **addresses** on the **network layer** |
| When generated / assigned ? | when sessions are initiated to communicate with server nodes | when sessions are initiated to communicate with server nodes |
| When disposed ? | when the sessions are closed | when the sessions are closed |

# Why we need Ephemeral Addresses?

## Because we have to enhance IP comm. security.

- We are ***sticking on***
  "**Legacy Concept of Address Usage**"
  (node utilizes ***only limited number*** of addresses).
- **Wide Address Space** can contribute to **security enhancements**
  - dynamically changing addresses
  - short life time addresses
  - mass-consuming addresses
  - etc.
- "**Ephemeral Address**" is not simple upgrading
  from port space to address space.
- "**Ephemeral Address**" is designed for **security enhancements**.

Let's **CHANGE** **Legacy Concept of Address Usage**.

**YES, we can**. (say together!)

# Comparison of "Ephemeral Addresses" and "Temporary Addresses" 1/2

In RFC4941, "Temporary Addresses" are defined
in order to enhance the privacy protection.

"Temporary Addresses" and "**Ephemeral Addresses**"
have the following similar functions.

1. They are used only for client nodes' source addresses.
2. They have lifetime, and theirs usable period is limited.
3. They can enhance the privacy protection.

· Goal is NOT to update "Temporary Address" spec.
Goal is to **CHANGE** **Legacy Concept of Address Usage**
for security enhancements.

# Comparison of "Ephemeral Addresses" and "Temporary Addresses" 2/2

| | Temporary Address | **Ephemeral Address** |
|---|---|---|
| Used for | **Multiple** Sessions | **Single** Session |
| Re-use Policy | **Re-used**<br>(weak from<br>security viewpoint) | **One** Shot / Disposal<br>***Never re-used***<br>(consume many addresses) |
| Address Lifetime | Rather long | Short (during the session) |
| Create / Dispose Timing | Vague | Crystal Clear |
| Design | Half-backed<br>Rather complex | Thoroughgoing Design<br>Very Simple |

# Concern Issues on Ephemeral Addresses

Q1: Is (64bit) Interface ID space really wide enough
for Ephemeral Address Usages ?

A1: Yes. **No Problems**!
(see the following ***quantitative analysis*** pages)

Q2: Which "Address Creation Rule" do we use?
A2: Out of scope for this I-D.
Let's start from "***at random creation***" rule.

Q3: How do we avoid DAD time consuming problem?
A3: Introduce new address state ("***Uncertain***" state)
(see next presentation on this issue)

# Quantitative Analysis: Let's calculate "Meet Again" Probability for the **same** Ephemeral Address

Condition:

Ephemeral Address Creation/Selection Rule is:
"***At Random***" from 64bit Interface ID space.

Probability Formula (Birthday Paradox):

"***n***" times probability:

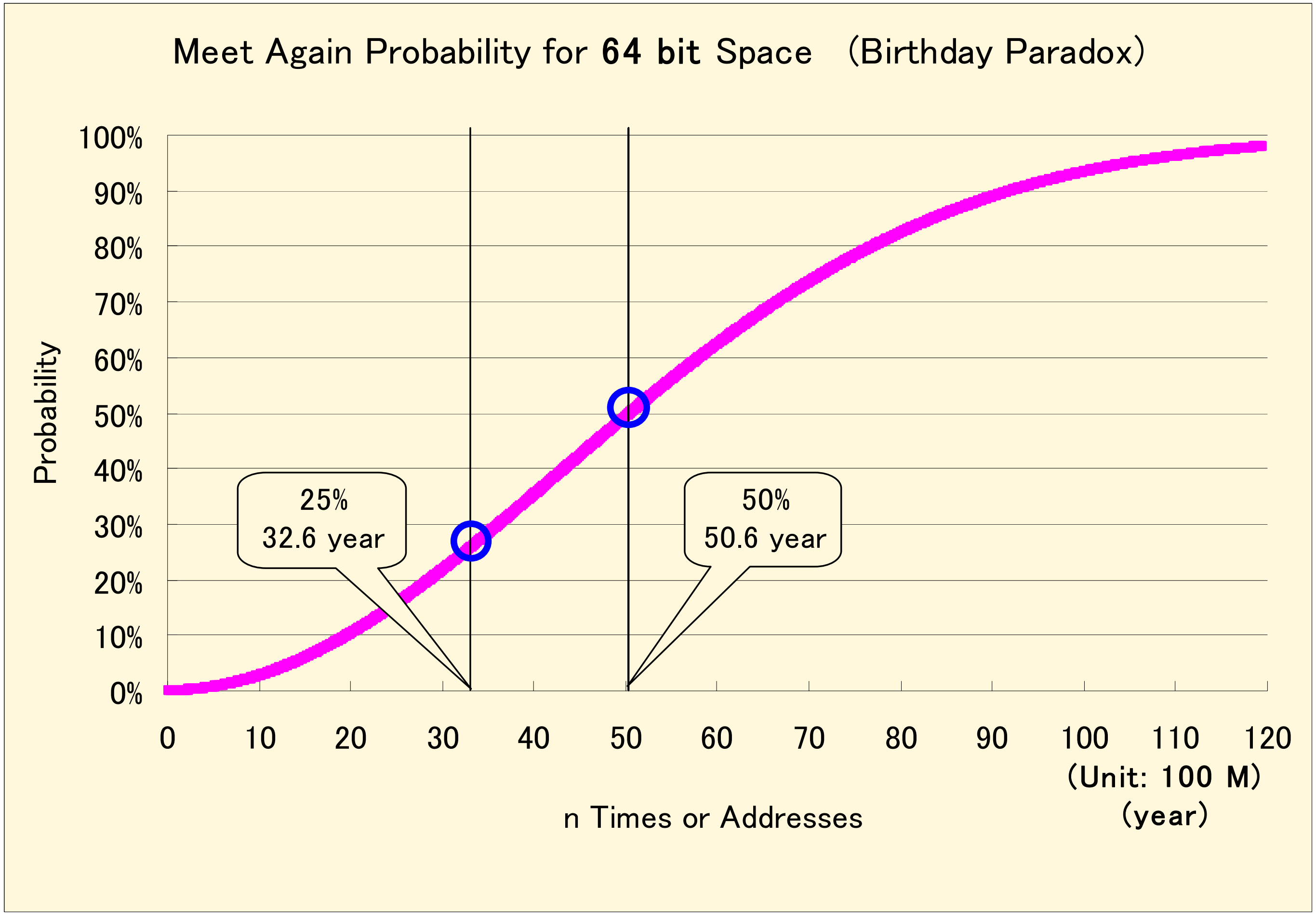
$$= 1 - (2^{64}-1)/2^{64} * (2^{64}-2)/2^{64} * \ldots * (2^{64}-\boldsymbol{n})/2^{64}$$

Estimation: Number of ***consumed addresses*** per (year, day, hour, min, sec)

| / year | / day | / hour | / min | / sec |
|---|---|---|---|---|
| 31,536,000 | 86,400 | 3,600 | 60 | 1.0 |
| 100,000,000 | 273,973 | 11,416 | 190 | 3.2 |

**"100M addr. / year"** is much enough (***sufficient estimation***)

# “Meet Again” Probability Results for the **same** Ephemeral Address




**Consume 100M addr. / year** (274k addr./day : 3.2 addr./sec)

10years: 2.8% 20years: 10.3%

**25%: 32.6 years** **50%: 50.6 years** 75%: 71.6 years

# Implementations

- “***Ephemeral Address***” specification has been implemented.
- Basic functionaries have been verified.

OS: FreeBSD6.2R (32bit / 64bit)

CPU: i386 / amd64

Since the spec. is simple,
it is easy to implement “Ephemeral Address.”

(If there are people who would like to implement “Ephemeral Address” on Linux or other OSs, please let us know.)

# Characteristics of Ephemeral Addresses

- No need to modify exiting applications
  (achieved by the ***kernel side modification only***)
- Only nodes who implement
  "Ephemeral Address" spec. get benefits.
- It may become difficult to administer clients' addresses
  - This is security enhancement technology.
  - New features (e.g., pseudonymity, unlinkability) may be brought, if you prepare good address creation rules.
- No problems are found.

# Next Step ?

- Update I-D
- Move to WG I-D ?

# Harmless IPv6 Address State Extension (**Uncertain** State)

<draft-kitamura-ipv6-uncertain-address-state-00.txt>


Hiroshi KITAMURA

NEC Corporation

kitamura@da.jp.nec.com

# Introduction and Goals

Propose a new IPv6 address state (“**Uncertain**”) as an extension of IPv6 address state specification.

Two Goals:

1. To achieve “**Address Reservation**” function.
2. To **avoid** a **DAD time consuming problem** for **dynamically created addresses** (e.g., **Ephemeral Addresses**, CoA of Mobile IPv6)

“**Uncertain**” address state is inserted between “Tentative” and “Valid” address states

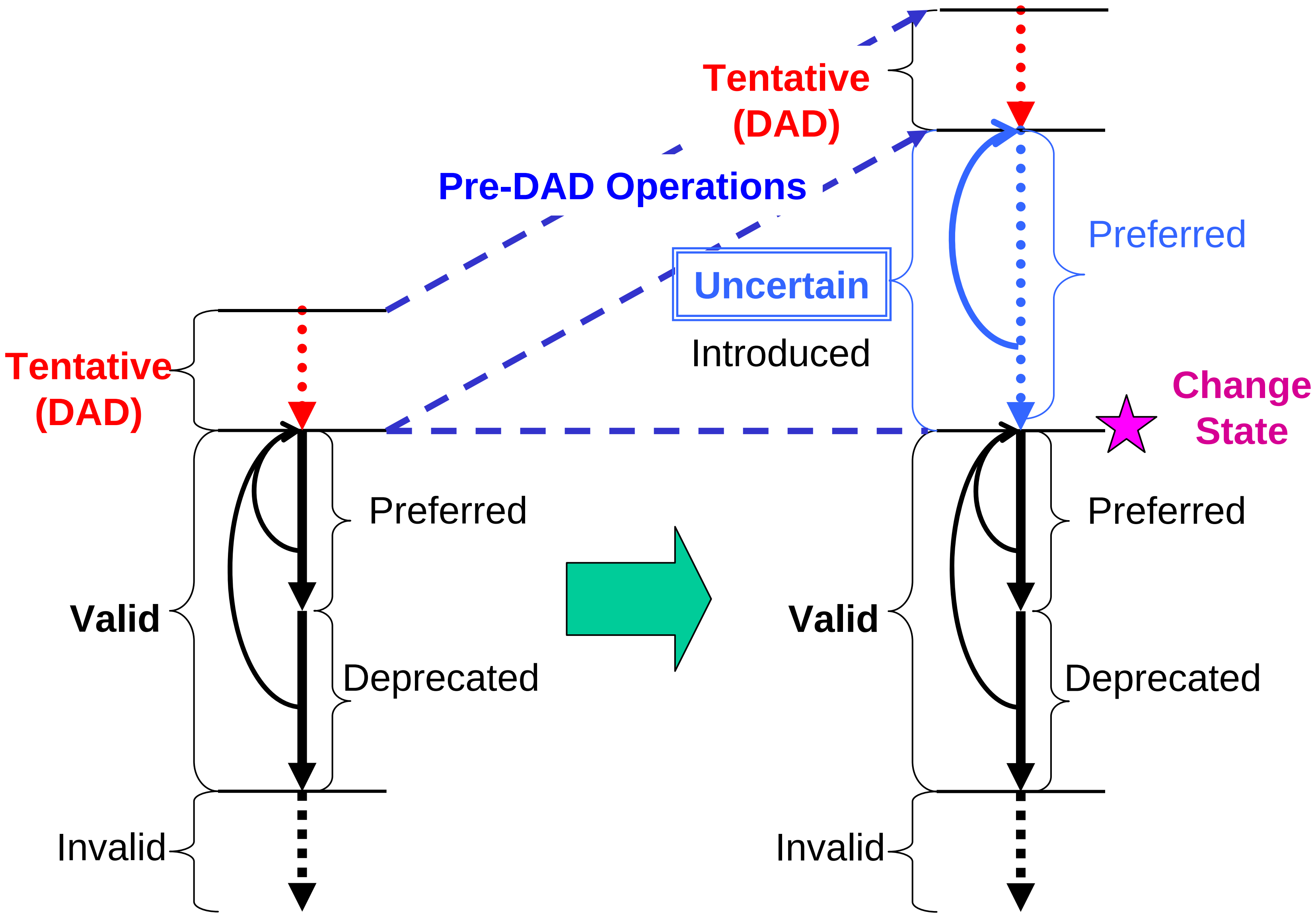
(“Tentative” -> “**Uncertain**” -> “Valid”)

# Design Policy:
# How to Avoid DAD time consumption

We do NOT choose "Optimistic" approach.

- **Do DAD** operations for **All** addresses
- But, DAD operations executing **timing is changed**
  - Address collision never happens
  - We don't have to worry about address collision cases.
  - No bad effects
    to the existing implementations are caused.

# Basic Design



# How to implement “Uncertain State” Focus on two types of NS messages

There are two types of NS messages

| | NS messages for DAD queries | NS messages for L2 Address queries |
|---|---|---|
| Source Address | unspecified address ( = ::) | **not** unspecified address (!= ::). |

These two messages are **distinguishable**.

# Implementation Design for "Uncertain State" Operations

| State \ NS | NS messages for DAD queries | NS messages for L2 Address queries |
|---|---|---|
| **Uncertain State** | Reply | **NOT** Reply |
| Valid State | Reply | Reply |
| *Function view* | **Reserve** / Own an address **exclusively**: The other nodes can NOT obtain the address | **NOT** Fill / Fill Neighbor Cache of the other nodes |

**Very simple Design:**

Only **NOT reply** to NS messages for L2 address queries

# “Uncertain State”, “Address Pool”, and Reserved Addresses

To implement “Uncertain State” is
almost same to implement “Address Pool”.

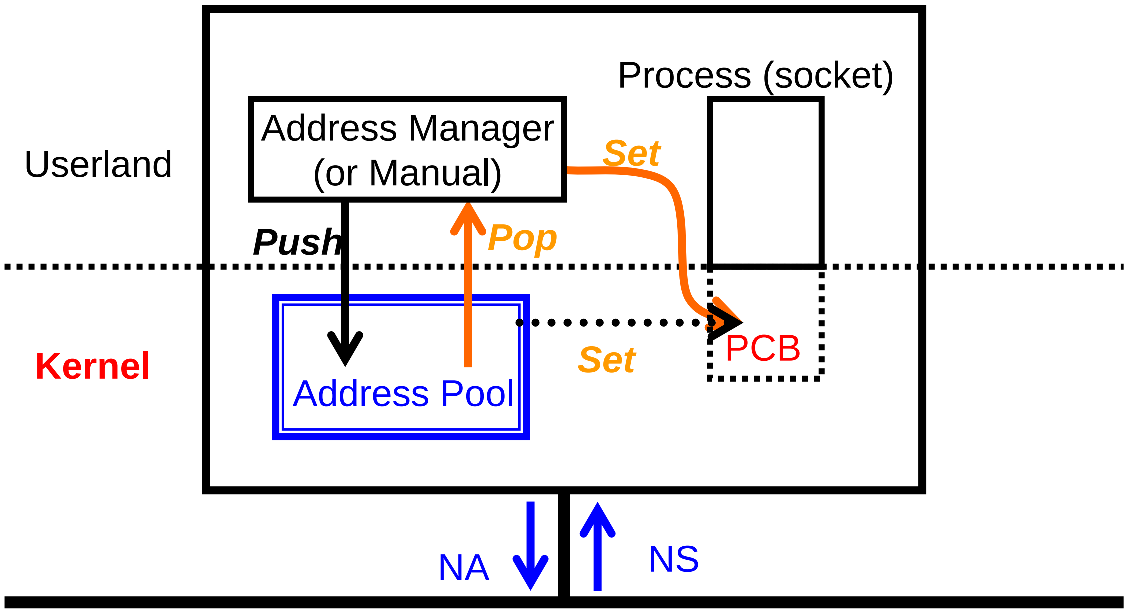
Reserved Addresses:

- They are stored in the Address Pool.
- Their address state is Uncertain address state.

When it becomes really necessary
for a node to utilize a reserved address:

- An address is taken from the Address Pool
- Its address state is ***changed into*** “**Valid**” address state ***without causing*** time consuming DAD operations.

# Address Pool and Address Manager




Address Pool is located in the kernel (like neighbor cache, routing table)

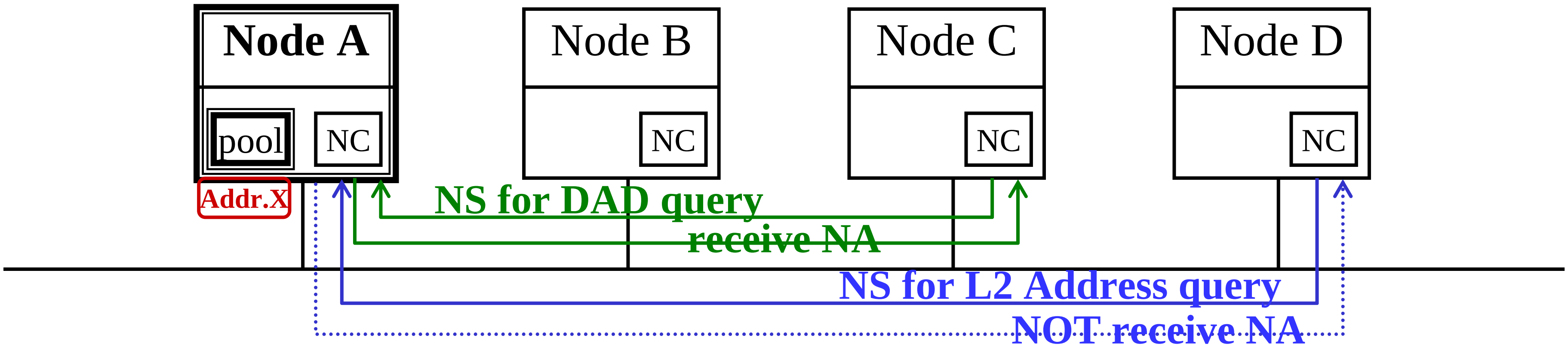
Uncertain Operations are implemented in the kernel

***Push:*** Save address(es) to the Address Pool

***Pop:*** Draw address(es) from the Address Pool

***Set:*** Set address to Process (socket)
[Actually, Set address info. to PCB]

# Network Environment (self pool) and Uncertain Operations



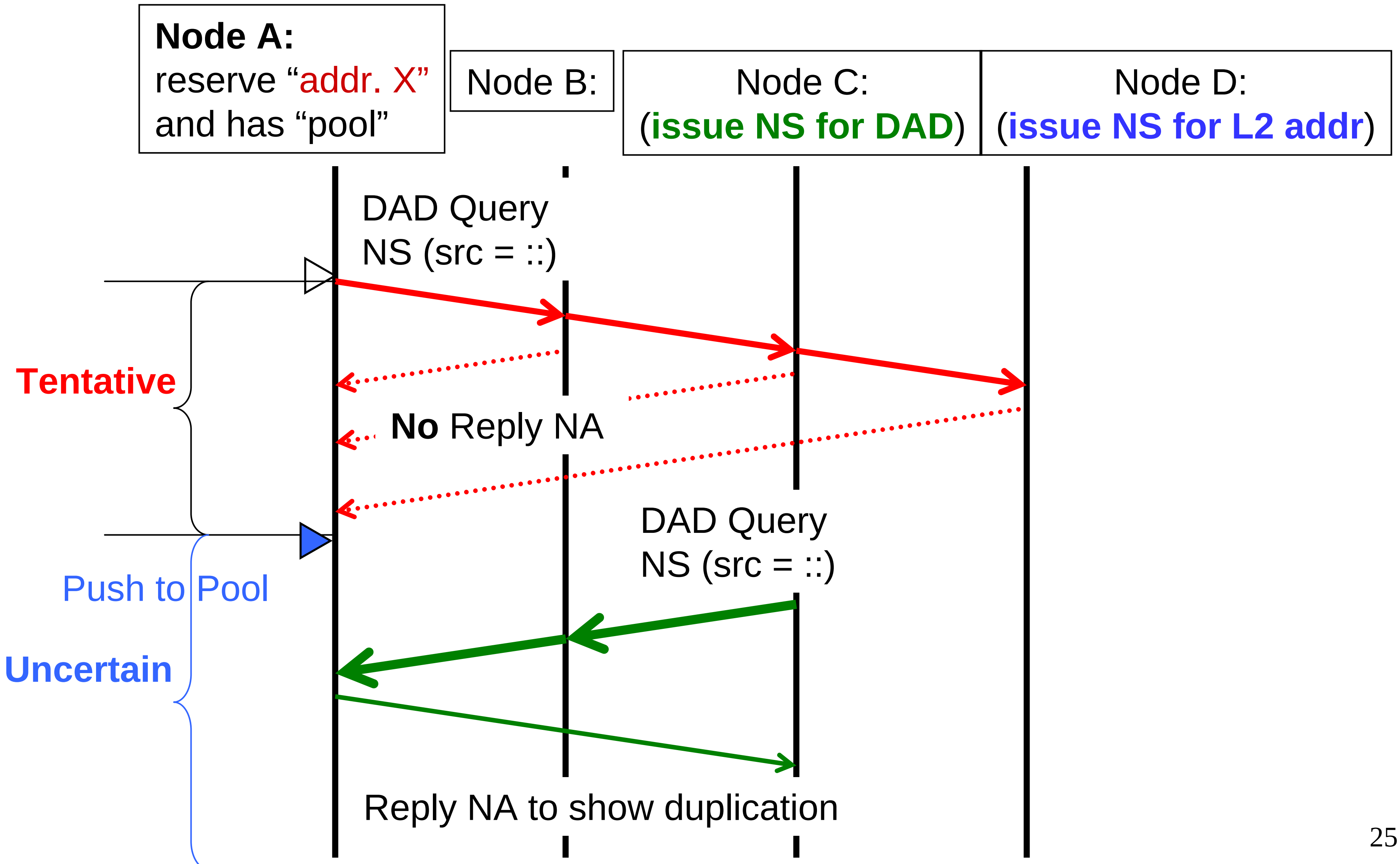

**Node A: Main player** (address consumer)
node who reserves "**addr. X**" and has "address pool"

Node B: [Simple neighbor node]

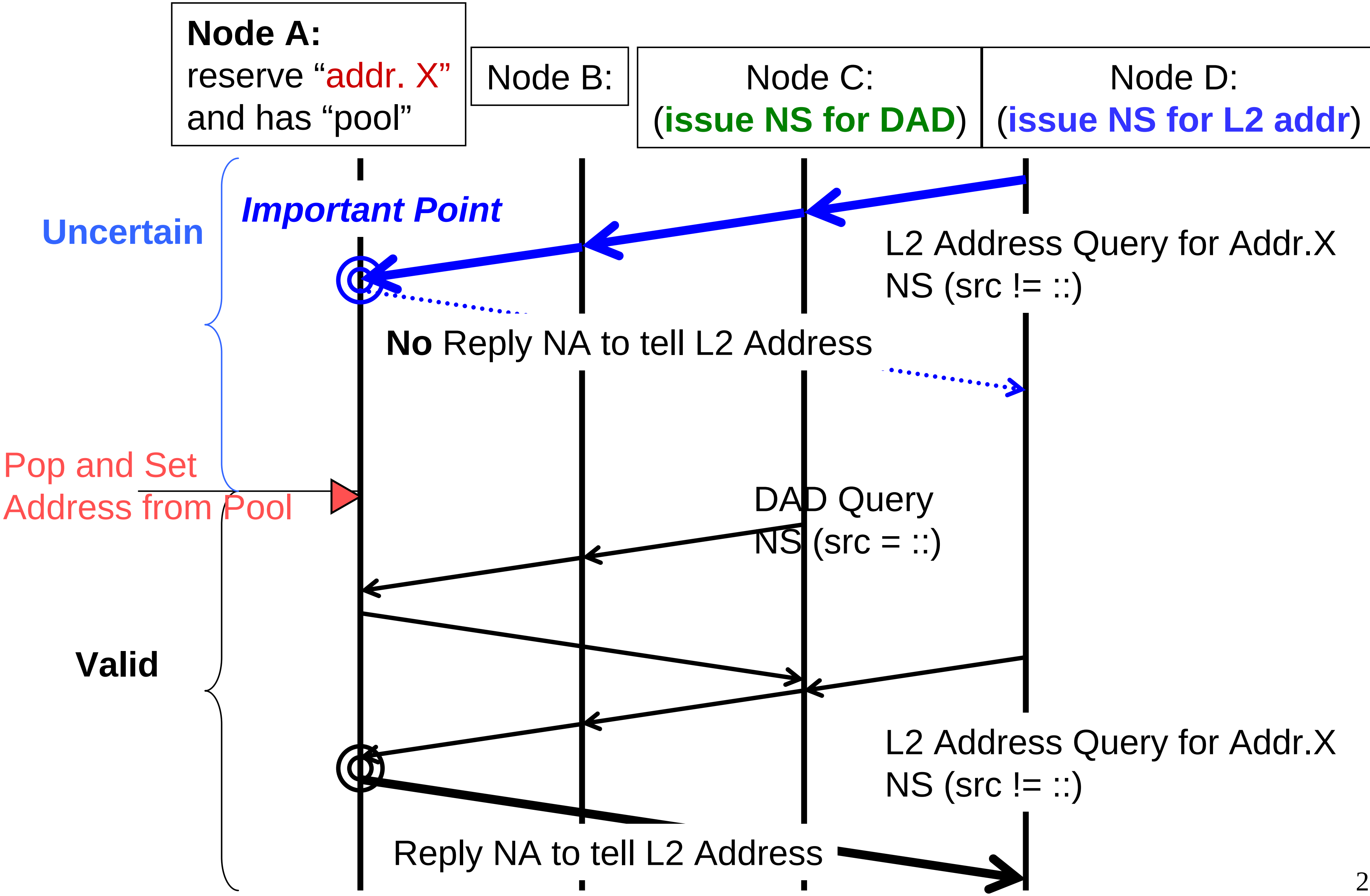
Node C: Node who wants to set/obtain "**addr. X**" **late**
(issues **NS for DAD query**, and receives NA)

Node D: Node who wants to talk with node who owns "**addr. X**"
(issues **NS for L2 Address query**, and **NOT** receives NA)

# Overview Sequences 1/2



# Overview Sequences 2/2



# Address Pool (Address Reserver) and Address Consumer 1/2

Two types are possible

- Self Pool Type:

  Address Reserver = Address Consumer

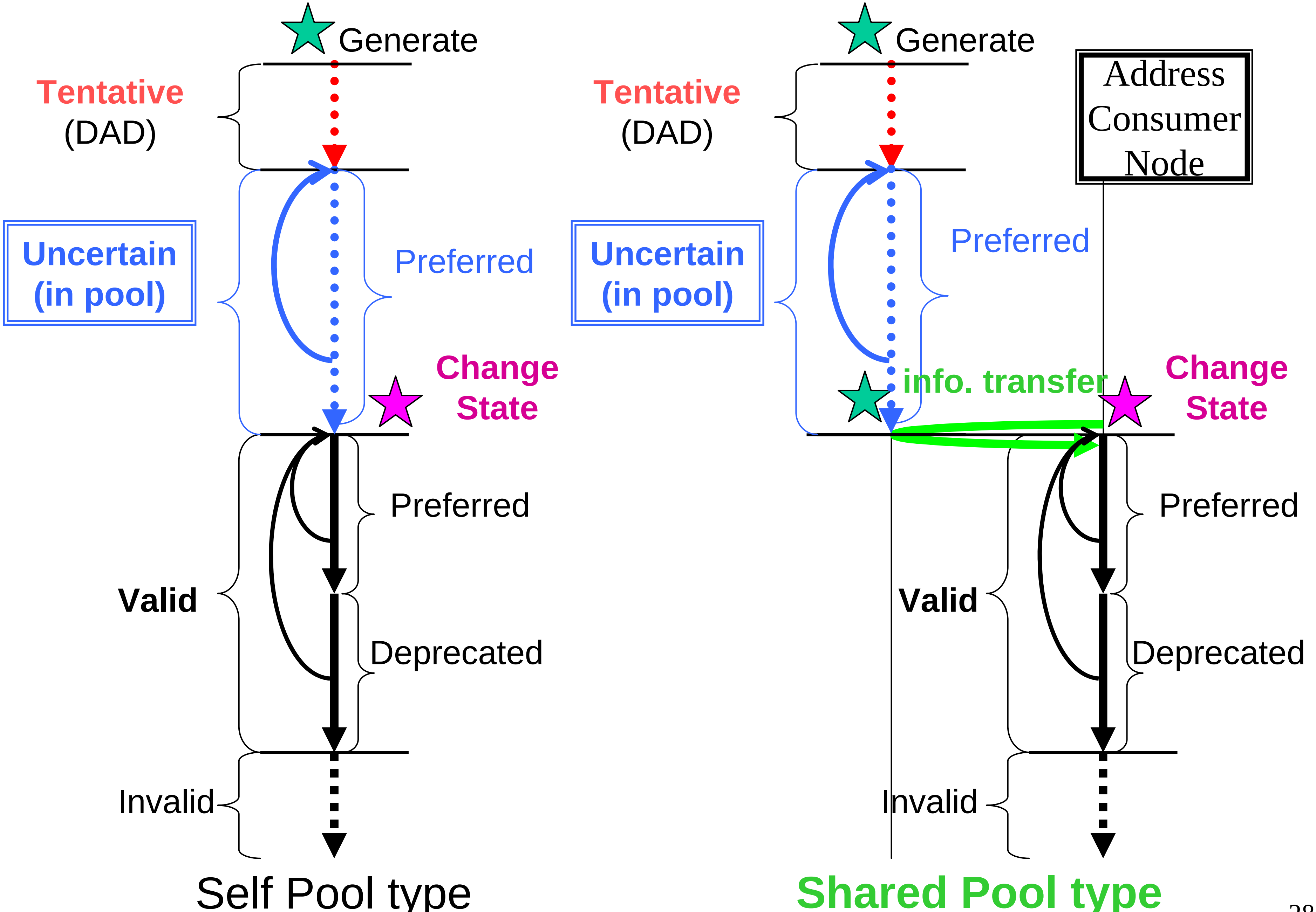
  (Simple: **described above**)

- **Shared Pool Type**:

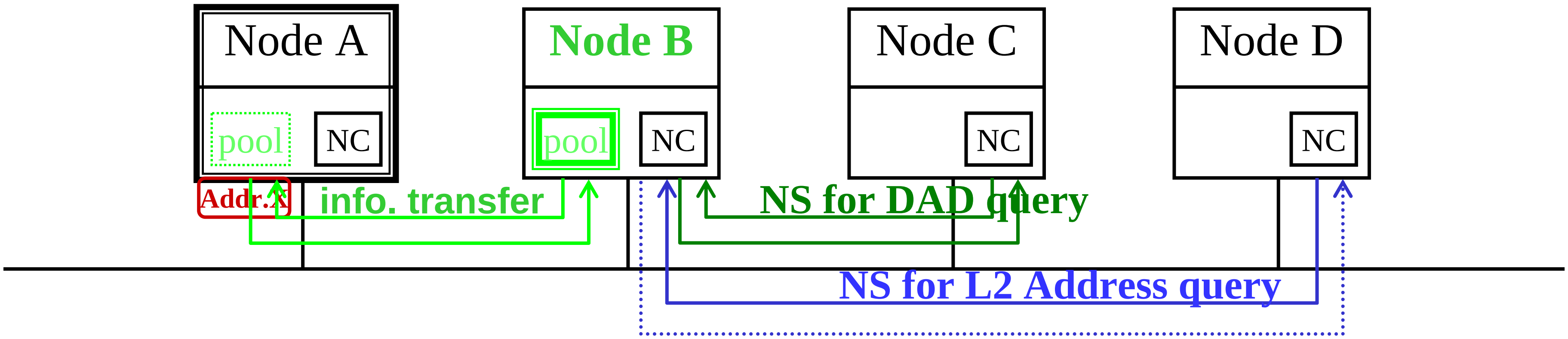
  Address Reserver **!=** Address Consumer

  (for Advanced Cases: to be used in future)

# Address Pool and Address Consumer 2/2



# Network Environment (shared pool) and Uncertain Operations




**Node A: Main player** (address consumer)
node who uses “**addr. X**” and may have “address pool”

**Node B: Pool Server** who has “**shared address pool**”

Node C: Node who wants to set/obtain “**addr. X**” **late**
(issues **NS for DAD query**, and receives NA)

Node D: Node who wants to talk with node who owns “**addr. X**”
(issues **NS for L2 Address query**, and **NOT** receives NA)

# Implementations

- “***Uncertain Address State***” specification has been implemented.
- Basic functionaries have been verified.

OS: FreeBSD6.2R (32bit / 64bit)

CPU: i386 / amd64

Since the spec. is simple, it is easy to implement “Uncertain Address State.”

(If there are people who would like to implement “Uncertain Address State” on Linux or other OSs, please let us know.)

# Uncertain State is Harmless Extension

- Uncertain address state is **harmless** extension
- It can **coexist** with current implementations without causing any problems

Because:

- –It is realized by NOT replying to NS messages for L2 address query.
- –NS messages are probing-type messages, they not to require mandatory NA replies.

# Harmless Feature Verification and Characteristics of Uncertain Address State

- "Harmless" feature have been verified with following rOSs.
  - FreeBSD 6.2 normal kernel
  - Linux 2.6.27-7 (Ubuntu 8.10)
  - MacOS X 10.3.9
  - Windows XP SP3, Windows Vista

- Only nodes who implement "Uncertain Address State" spec. get benefits.
- No problems are found

# Next Step ?

- Update I-D
- Move to WG I-D ?