Block Address Registration and Claiming (BARC)

Roger Marks
EthAirNet Associates
2021-03-12

Acknowledgements

thanks to Antonio de la Oliva
for review and constructive comments

thanks to other 802.1 WG and 1722 WG participants for constructive discussions during earlier comment resolution, particularly to Mick Seaman, Norm Finn, Don Pannell, and Max Turner

Introduction

- P802.1CQ/D0.5 was reviewed in Task Group Ballot.
- Comment resolution indicated that significant changes were required.
- This contribution summarizes an approach to the next draft.
- Approach is based on address blocks, as raised during comment resolution.
- Each Address Block (AB) is identified by an Address Block Identifier (ABI).
- Some ABIs (CABIs) are designated for self-claiming.
- Other ABIs (RABIs) are designated for management by a Registrar.
- Address Registration and Claiming (ARC) protocol is outlined here, with:
  - Block Address Registration and Claiming (BARC)
  - Address-Range (AR) claiming, incorporating MAAP from IEEE 1722, per comment resolution.
    - not using any BARC addresses

BARC assigns MAC Addresses in Address Blocks

1) Address Blocks (ABs) include local addresses.

2) An AB includes both unicast and multicast address subblocks.

3) No address falls within more than one AB.

4) An AB Identifier (ABI) identifies each AB.

5) Registrars hold multi-blocks sets of ABs, identified by multi-block identifiers (MBIs).

6) An ABI within an MBI is a registrable ABI (RABI).
   - identifies Registrable Address Blocks (RABs) holding Registrable Addresses (RAs)

7) An ABI not in an MBI is a Claimable ABI, claimable by a Claimant without a Registrar.
   - identifies Claimable Address Blocks (CABs) holding Claimable Addresses (CAs)

8) Each ABI and MBI is a multicast address and not in any AB.

9) A large set of Temporary Unicast Addresses (TUAs) is specified
   - for initial discovery by Claimant lacking a unicast address.

MAC Address Categorization

determinable via inspection:	**Expanded name**	**Role**	**I/G**	**indicates, by inspection**
not specified by BARC				
CA	claimable address, in claimable address block (CAB)	BA	U,M	CABI, CABI type, CAB (including all other CAs in CAB)
CABI	CAB identifier	BI	M	CABI type, CAB
MBI	multi-block identifier	MBI	M	MBI type, RABI type, RABIs
RA	registrable address, in registrable address block (RAB)	BA	U,M	RABI, RABI type, MBI, MBI type, all other RAs in RAB
RABI	RAB identifier	BI	M	RABI type, MBI, MBI type, RAB (including all other RAs in RAB)
RABIA	(unicast) RABI address (one of many)	BI	U	RABI
TUA	temporary unicast address		U	note: ~6.8E10 to choose among

BA= block address ; BI = block identifier

BARC MAC Address Structure

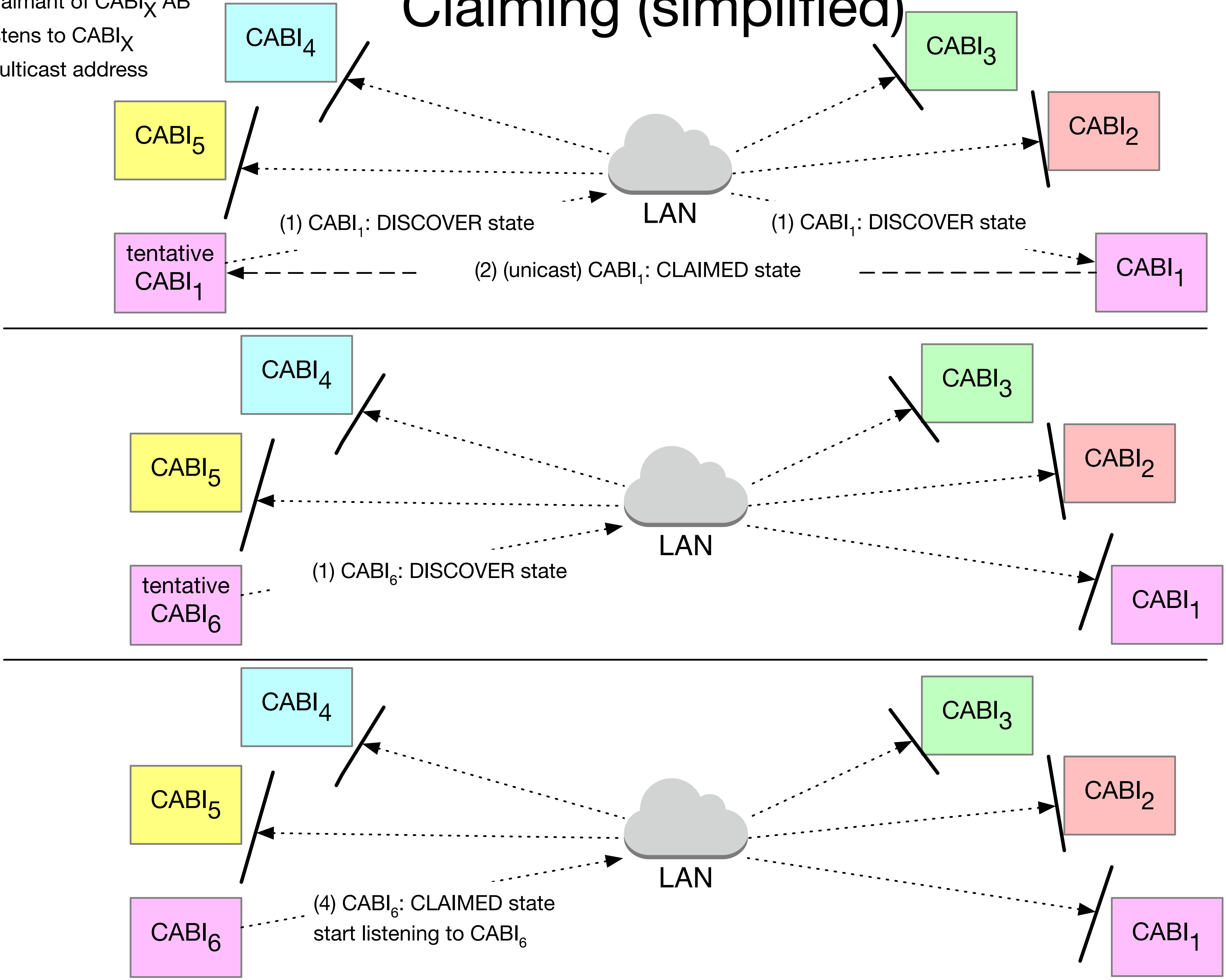
N11	0	always 0 (remaining 15 values for non-BARC use)
N10	1 1 1 M	M is the usual multicast (I/G) bit; 111 is local “SAI” range per IEEE Std 802c
N9	p r j k	address block includes subblocks of 16^{jk} contiguous addresses
N8	n a b c	multi-block includes 16^{abc} address blocks (N8 unstructured in CABI)
N7		
N6		
N5		
N6		
N5		
N2		
N1		
N0		

12 nibbles per 48-bit address

- M, p, r, and n bits distinguish Block Addresses (BAs) and Block Identifiers (BIs)
    - e.g. CABI has M=1, p=0, r=0
- see Appendix for details

Claiming (simplified)

Claimant of CABI$_X$ AB listens to CABI$_X$ multicast address



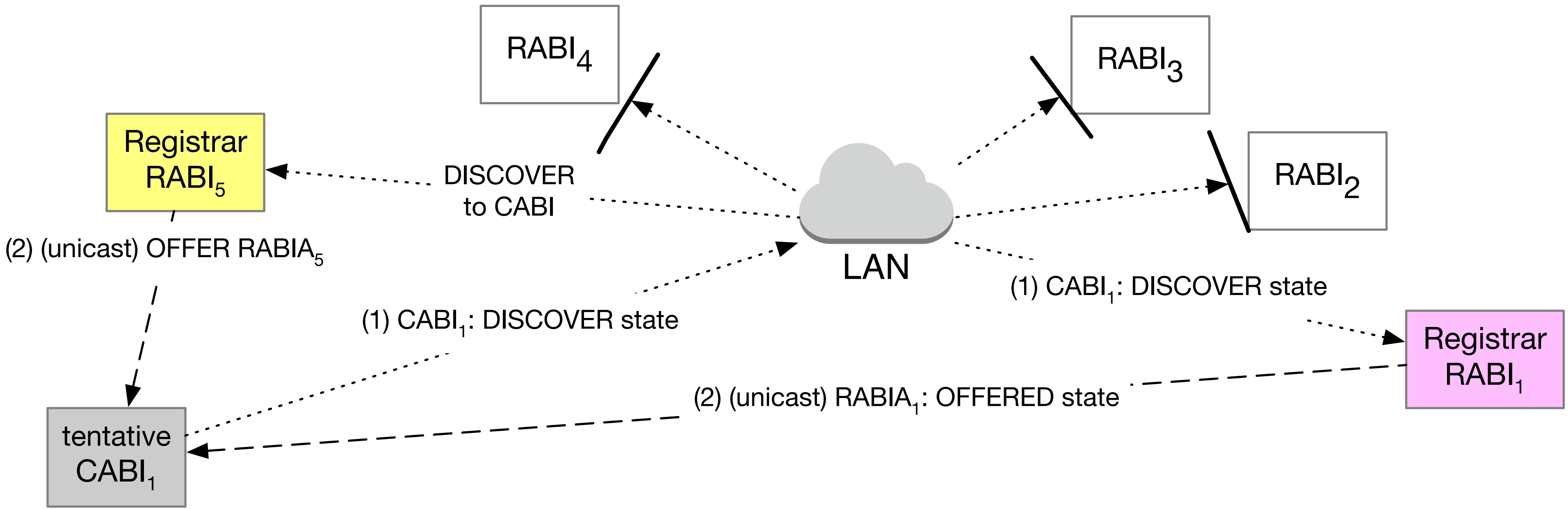
Registrar

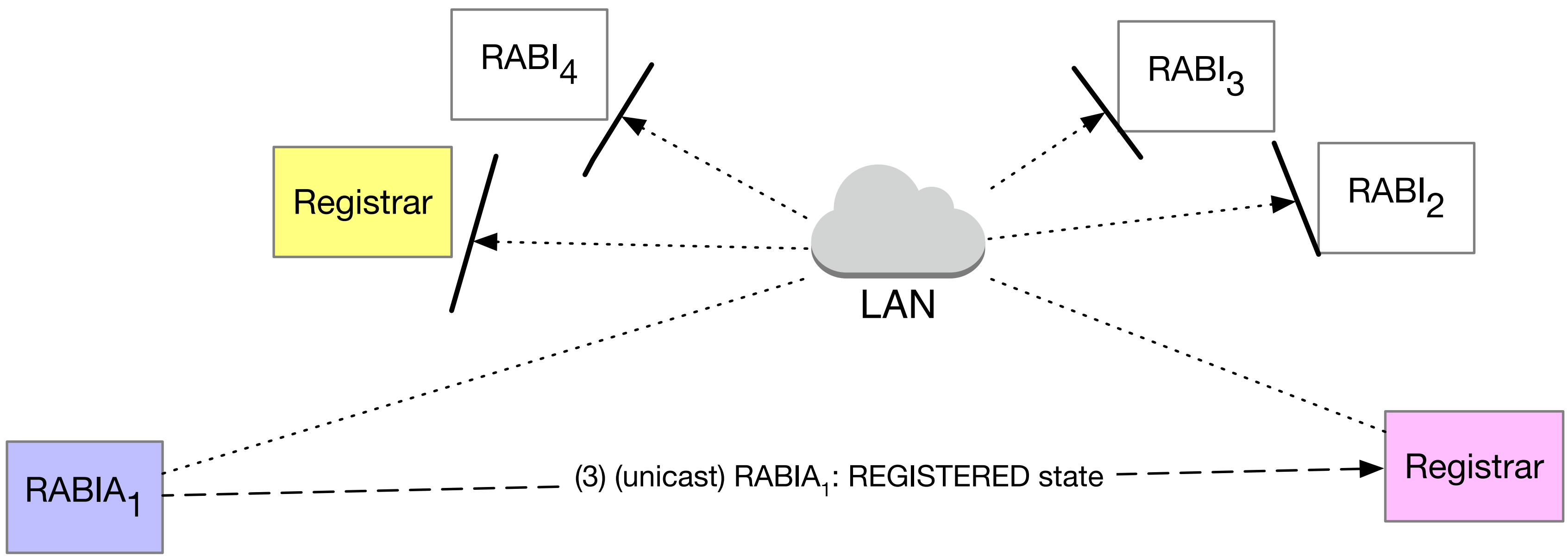
- Registrar maintains an inventory of RABIs (within MBIs).
  - a protocol specifies how Registrars acquire MBIs.
  - set of RABIs is disjoint from the set of CABIs
    - AB is either claimable (CABI) or registrable (RABI); not both

- Registrar listens for all messages to a CABI.
  - M=1, p=0, r=0, i.e. DA begins 0000-1111-00
    - [MMRP NumberOfValues field is 13 bits]
- Registrar can respond to a DISCOVER with an OFFER of an AB in its inventory.
  - The OFFER defends the DISCOVER message's CABI.
  - Client claims an offered RABI, similar to claiming CABI.
    - Registrar does not assign RABI but tracks its registration.
  - OFFER cites one of the RABI's RABIA, not the RABI directly.
    - claim is then sent to Register at the unicast RABIA
    - has some advantages

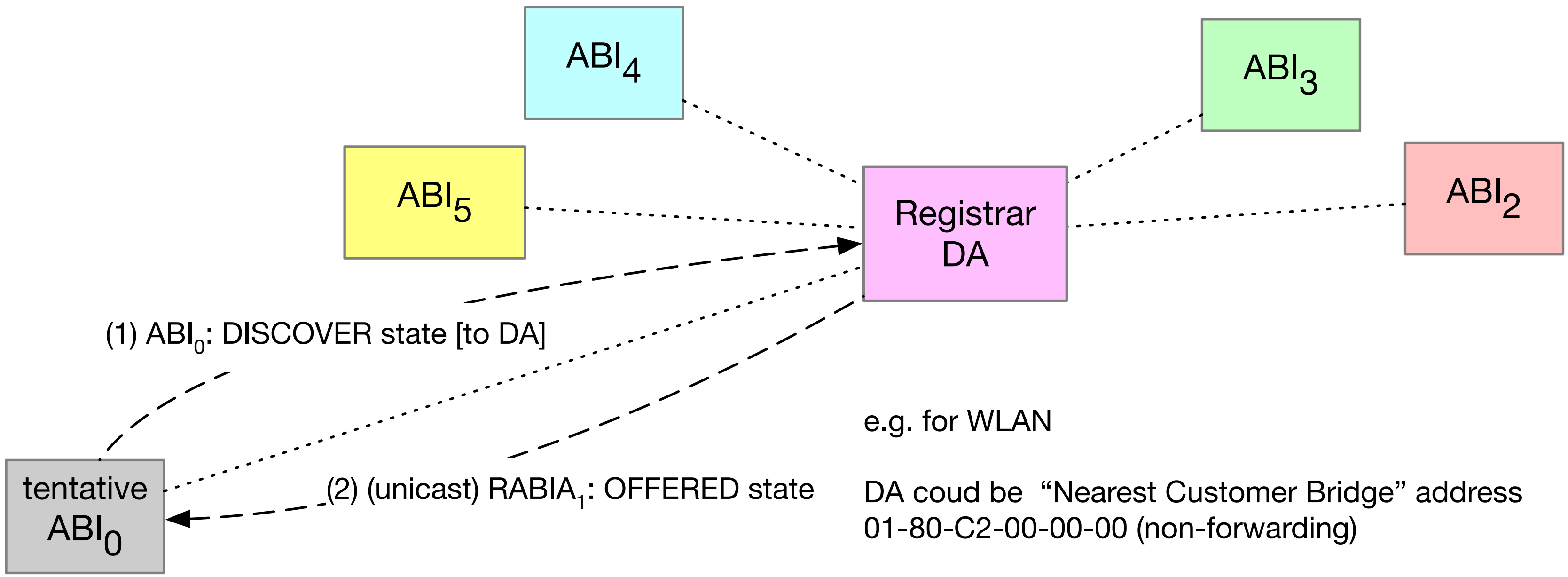
- *Claimant need not be aware of Registrar when initiating a claim.*

Operation with Registrars



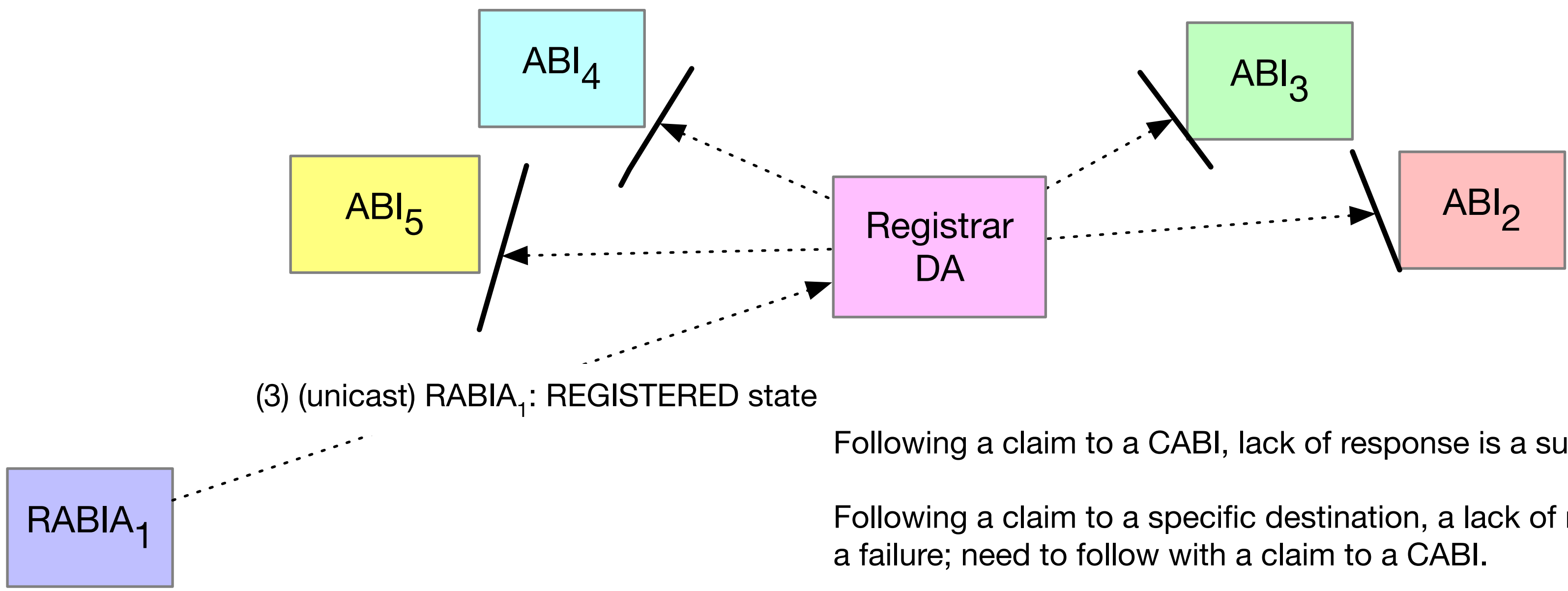


if Registrar expected at specific address



e.g. for WLAN

DA coud be “Nearest Customer Bridge” address
01-80-C2-00-00-00 (non-forwarding)



Following a claim to a CABI, lack of response is a success.

Following a claim to a specific destination, a lack of response is
a failure; need to follow with a claim to a CABI.

BARC Design

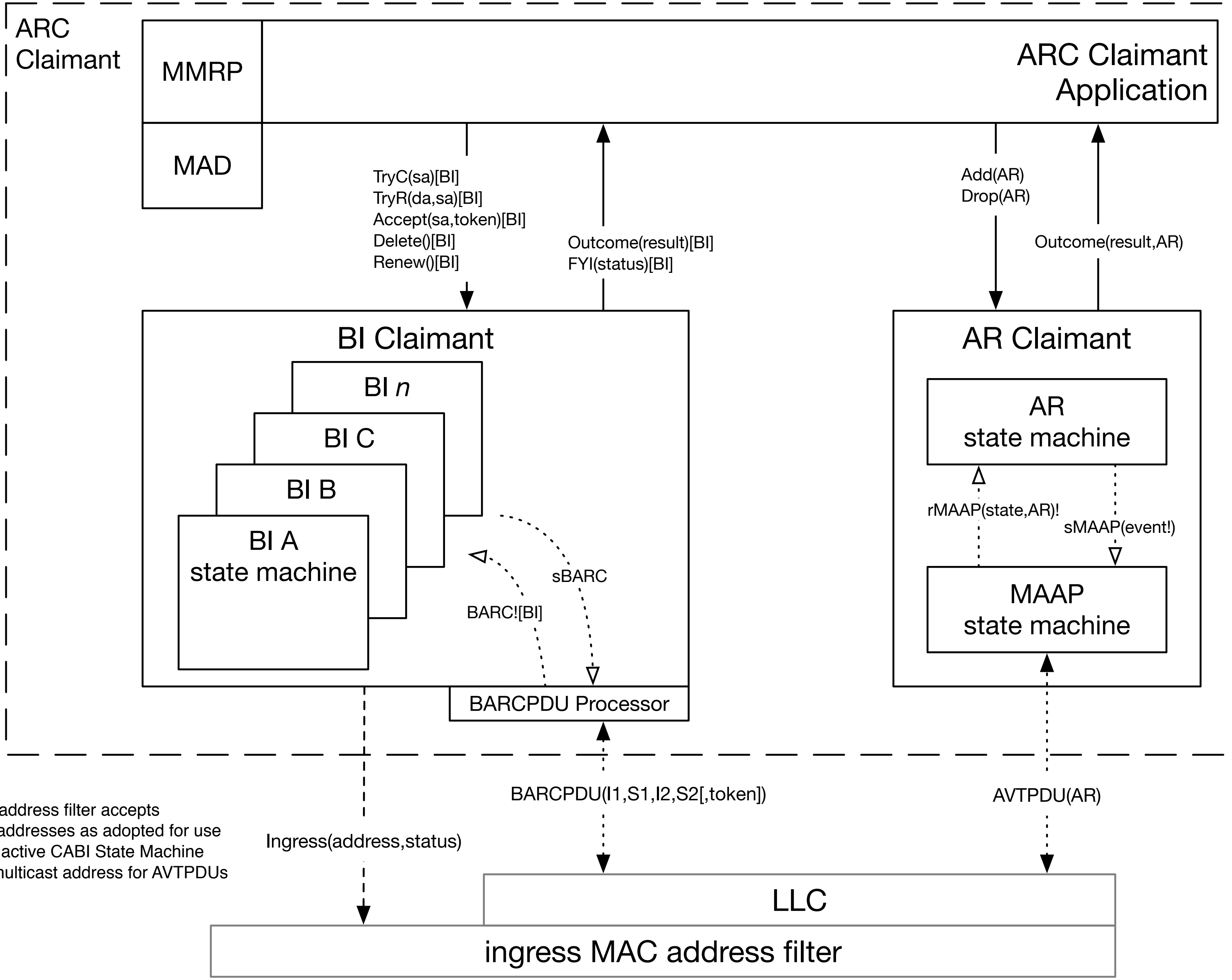
- A BARC architecture follows, with details including state machines.
  - additional details in Appendix

- Listened carefully to comments and discussion raised in P802.1CQ/D0.5 TG Ballot

- BARC (Block Address Registration and Claiming) is put into the broader context of Address Registration and Claiming (ARC), which supports both:
  - address blocks (ABs), identified by Address Block Identifiers (ABIs)
  - address ranges (ARs), excluding addresses specified by BARC

- ARC is the general protocol
  - BARC handles BI Registration and Claiming
  - existing MAAP handles AR Claiming

ARC Architecture – ARC Claimant



ingress MAC address filter accepts

- unicast addresses as adopted for use
- BI of an active CABI State Machine
- MAAP multicast address for AVTPDUs

BARCPDU Summary

field name	purpose	content
DA	dest addresss	DA
SA	source address	(TUA allowed if S=D or A)
E	Ethertype	[tbd; could be 22F0 (MAAP Ethertype)]
t	subtype	[tbd, per 1722 WG; see IEEE 1722 Table 6]
S1	State	D, C, V, R, T, MD, MC, N(null)
I1	identifier	(an address)
S2	State	O, S, N(null)
I2	identifier	
T	token	

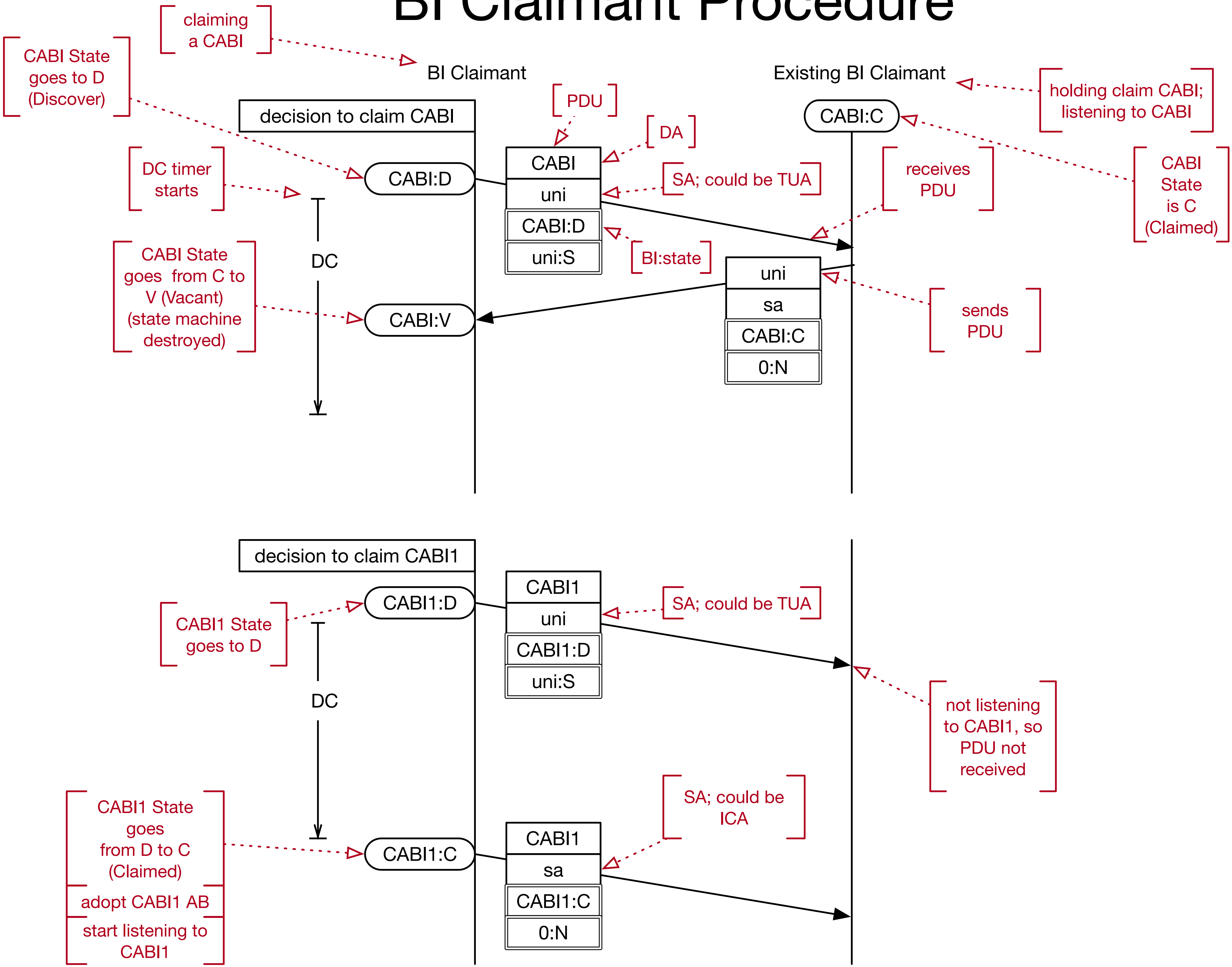
AVTPDU Summary

field name	purpose	content
DA	dest addresss	91:E0:F0:00:FF:00 for MAAP multicast
E	Ethertype	22F0 (MAAP Ethertype)
t	subtype	FE per IEEE 1722 Table 6

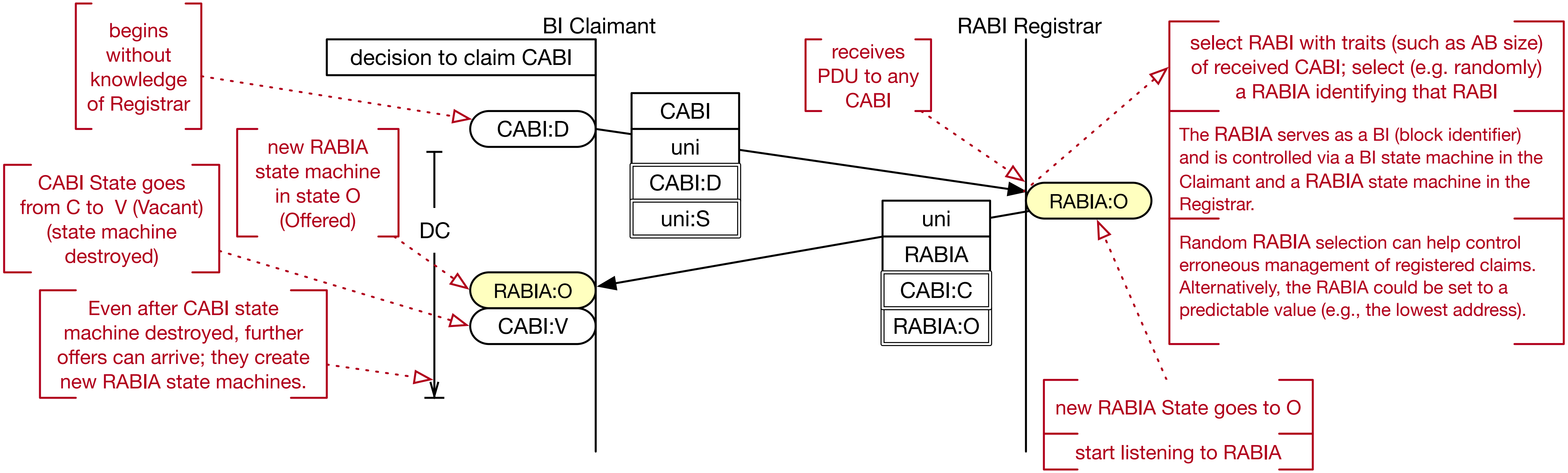
ABI Claimant – BI State Transition Table

	State (BI/CABI)			State (BI/RABIA)	
Event	**VACANT (V)**	**DISCOVERY (D)**	**CLAIMED (C)**	**OFFERED (O)**	**REGISTERED (R)**
TryT(da,sa)!	sBARC(BI:T,sa:S){da,sa} Start DiscoverT_Timer **DISCOVERY**				
DiscoverT_Timer!		Outcome(T) **VACANT**			
BARC(O)!	FYI(Offer) StartOffer_Timer **OFFERED**				
Offer_Timer!				**VACANT**	
Accept(sa,token)!				Ingress(sa,pass) R_sa==sa R_token==token sBARC(BI:R,sa:S,token){BI,sa} Start Renew_Timer(R) **REGISTERED**	
BARC(R,sa,token)!					if sa= R_sa and token= R_token then Start Renew_Timer(R)
BARC(V,sa,token)!					if sa= R_sa and token= R_token then Ingress(sa,filter) FYI(Revoked) **VACANT**
TryC(sa)!	sBARC(BI:D,sa:S){BI,sa} Start DiscoverC_Timer **DISCOVERY**				
DiscoverC_Timer!		Ingress(BI,pass) Outcome(C) sBARC(BI:C,0:N){BI,sa} Start Renew_Timer(C) **CLAIMED**			
BARC(C)!		Outcome(V) **VACANT**	sBARC(BI:C,0:N){BI} FYI(Alert)		
BARC(D,SA)!		Outcome(V) **VACANT**	da=SA sBARC(BI:C,0:N){da}		
Delete()!		**VACANT**	sBARC(BI:V,0:N){BI} Ingress(BI,filter) **VACANT**		sBARC(BI:V, R_sa,R_token){BI,R_sa} Ingress(BI,R_sa) **VACANT**
Renew_Timer! or Renew()!			FYI(Renewed) sBARC(BI:C,0:N){BI,sa} Start Renew_Timer(C)		FYI(Renewed) sBARC(BI:R,sa:S,token){BI,sa} Start Renew_Timer(R)

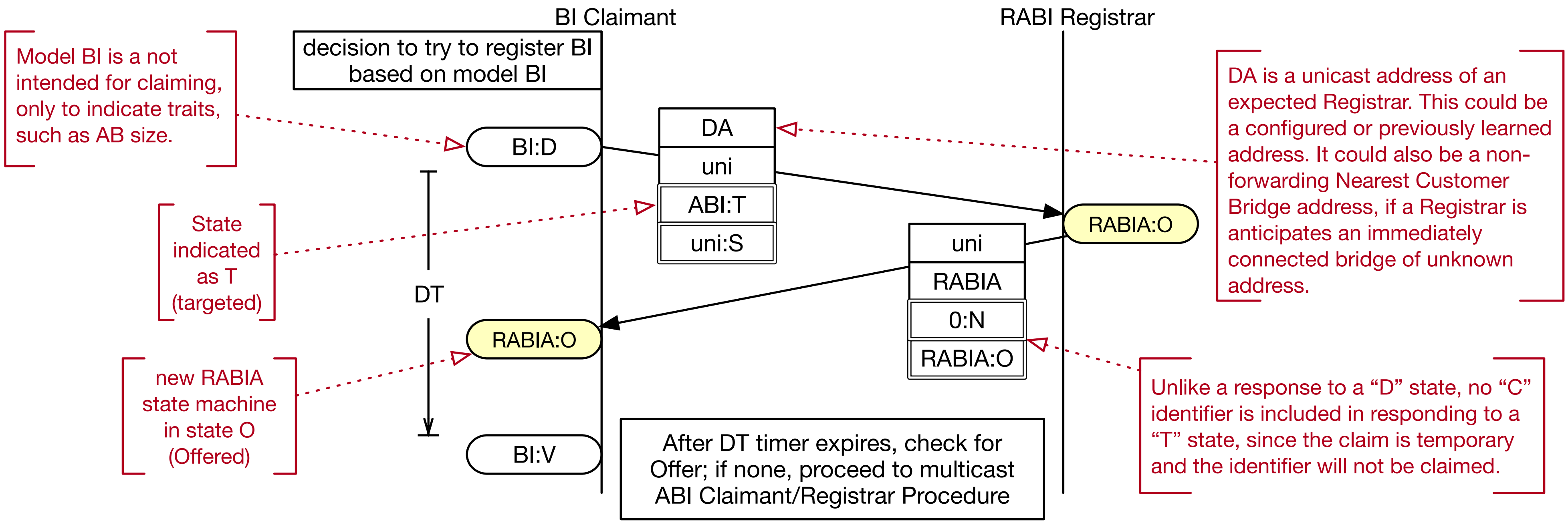
BI Claimant Procedure



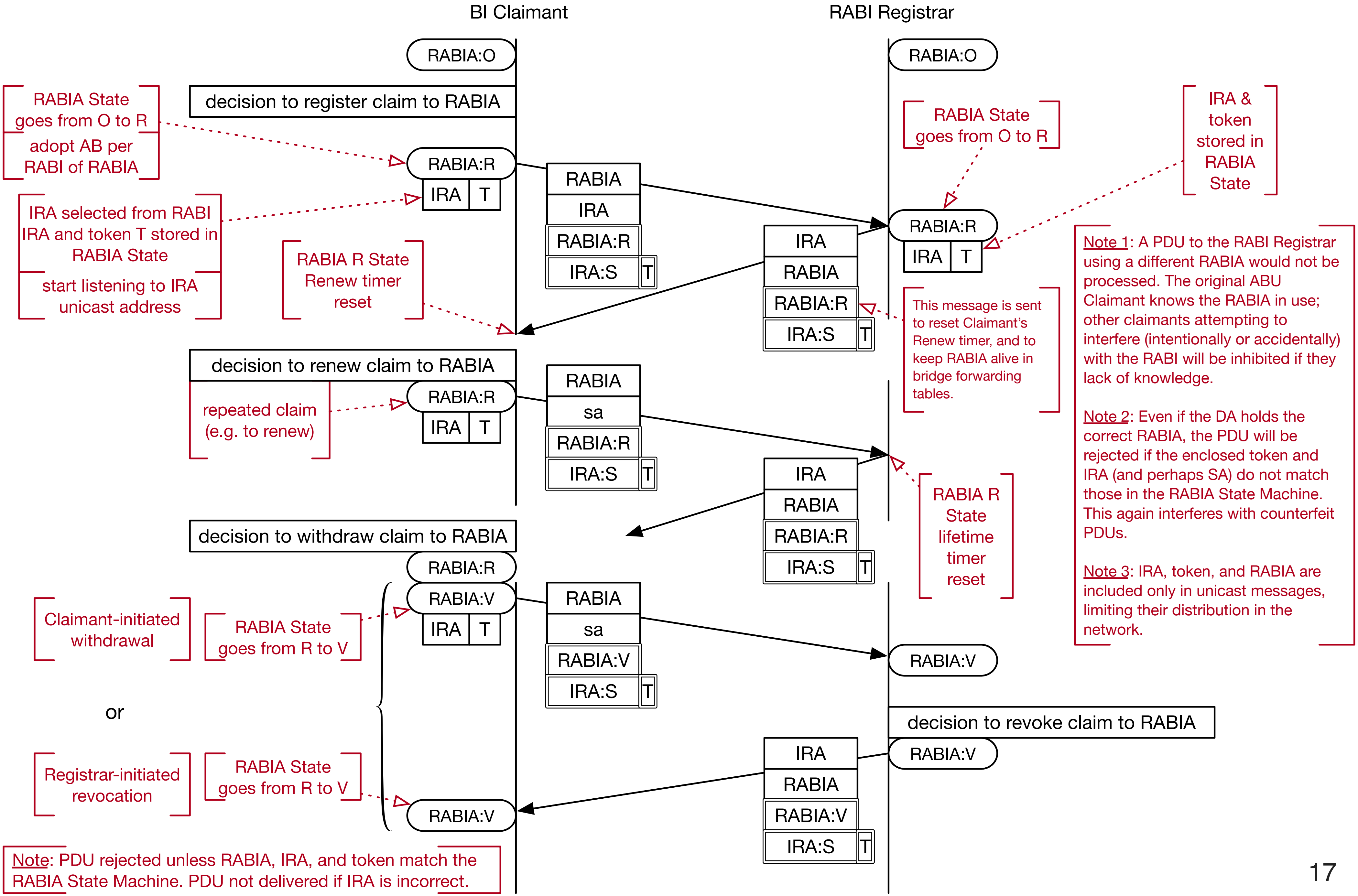
BI Claimant/Registrar Procedure: Multicast



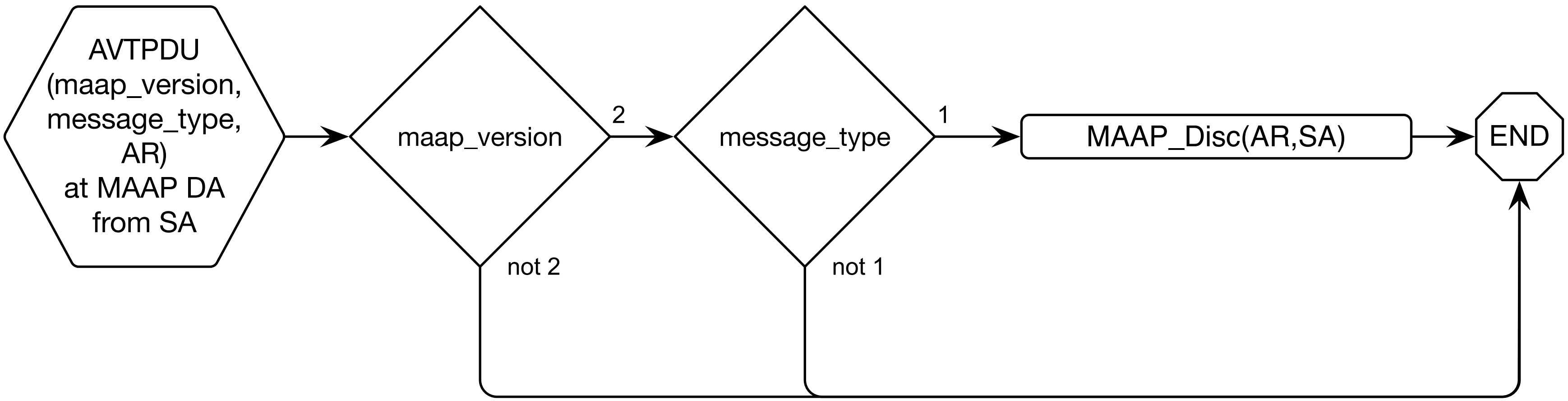
BI Claimant/Registrar Procedure: Targeted



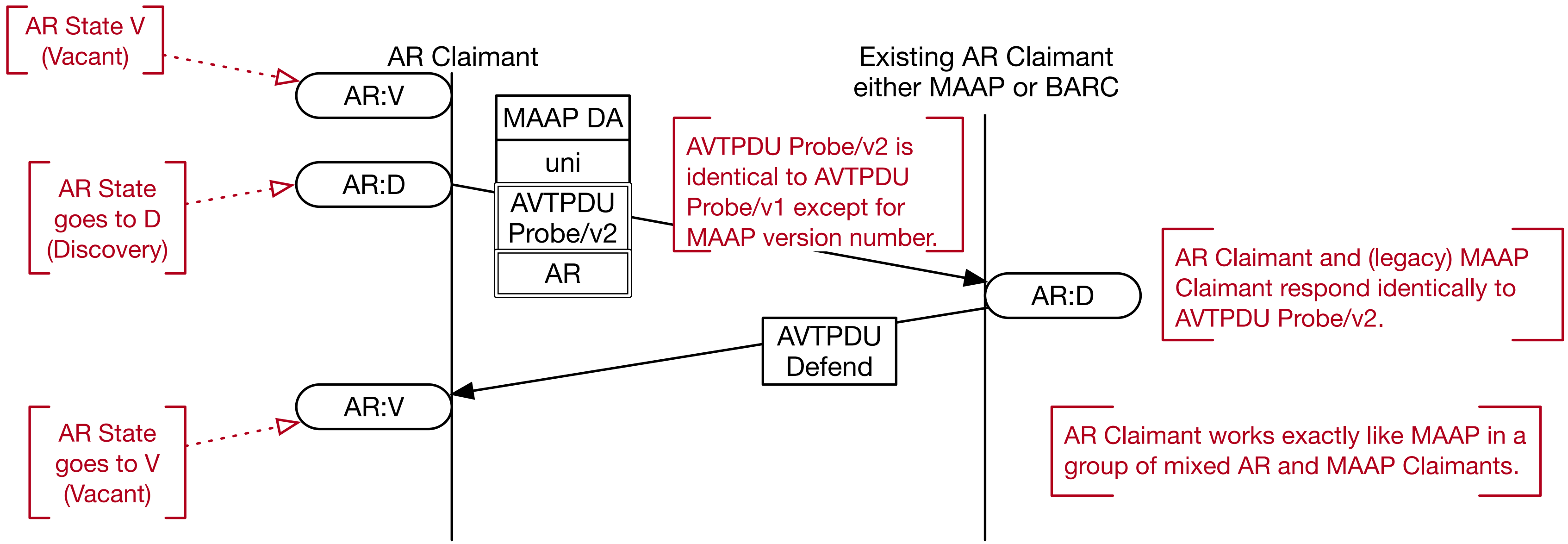
Registration, Renewal, and Withdrawal of an Offer

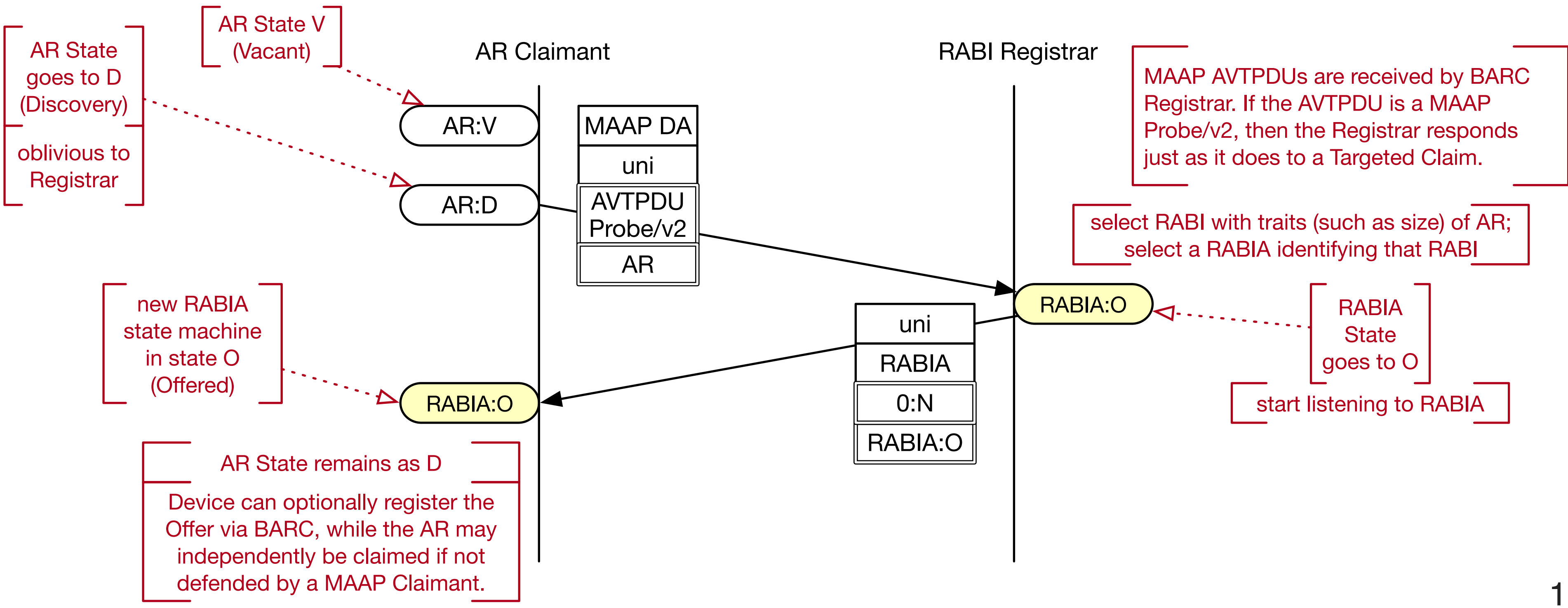


BARC Registrar: AVTPDU Processor



AR Claimant Procedure





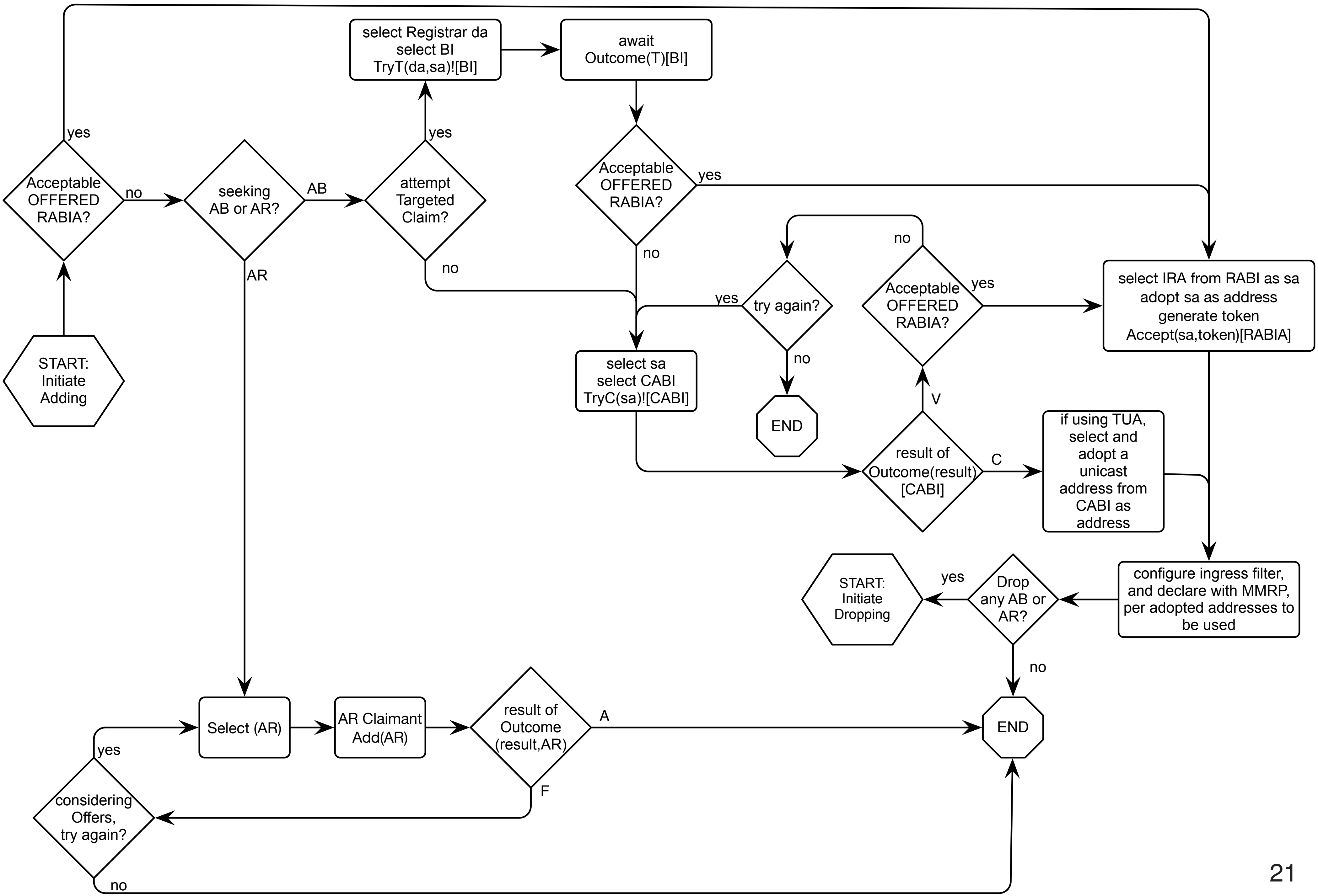
AR State Transition Table

	State		
Event	**VACANT (V)**	**DISCOVERY (D)**	**ACQUIRED (A)**
Add(AR)!	sMAAP(Begin(AR)!) **DISCOVERY**		
rMAAP(AR:Defend)!		Outcome(A,AR) **ACQUIRED**	
rMAAP(AR:Initial)!		Outcome(F,AR) **VACANT**	Outcome(X)[AR] **VACANT**

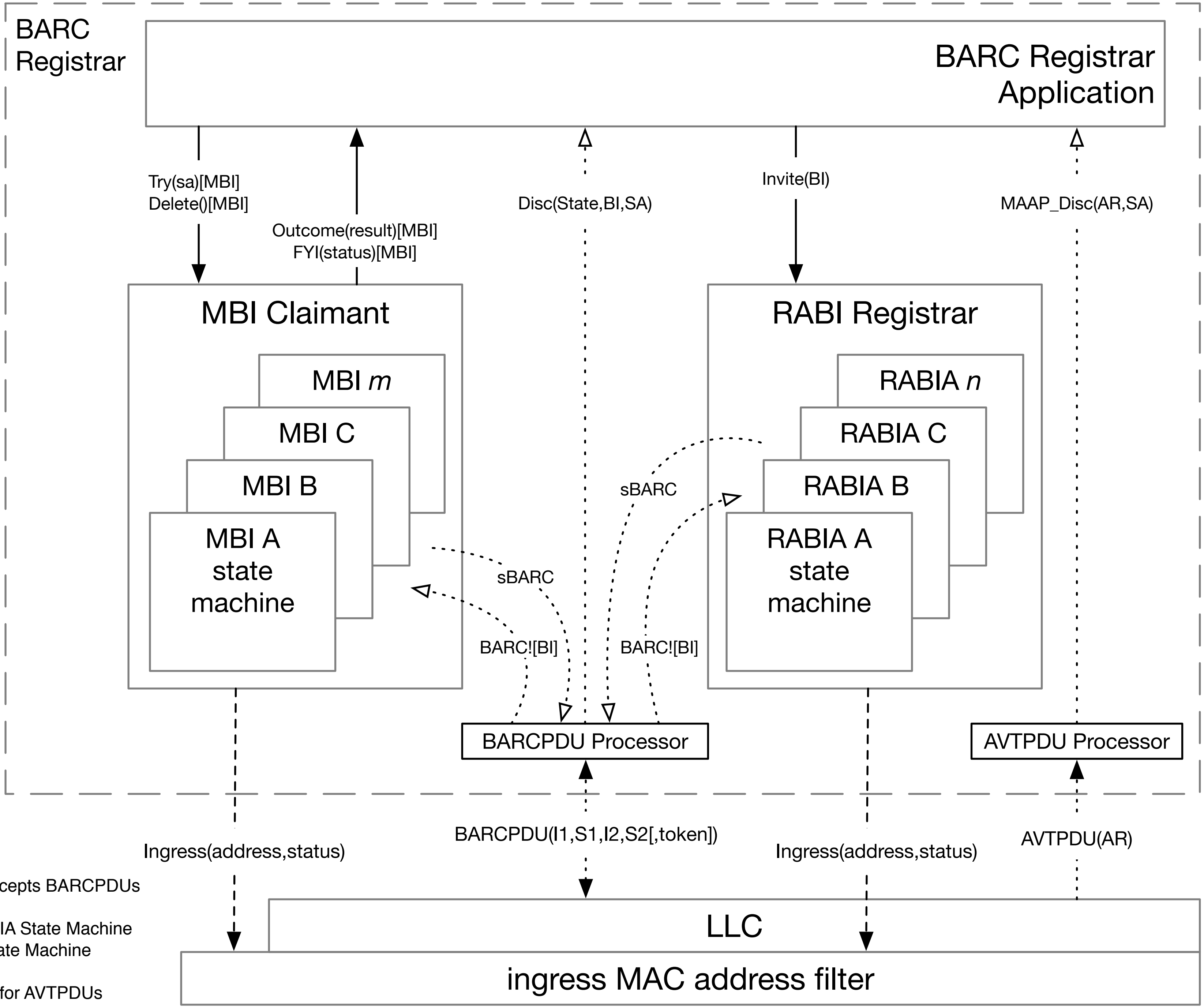
rMAAP(AR:State!) invokes an event at the state machine when the MAAP state changes to State

sMAAP(Action!) invokes Action! event at MAAP state machine

ARC Claimant Application Process: Add Claim



BARC Architecture – Registrar



ingress MAC address filter accepts BARCPDUs addressed to:

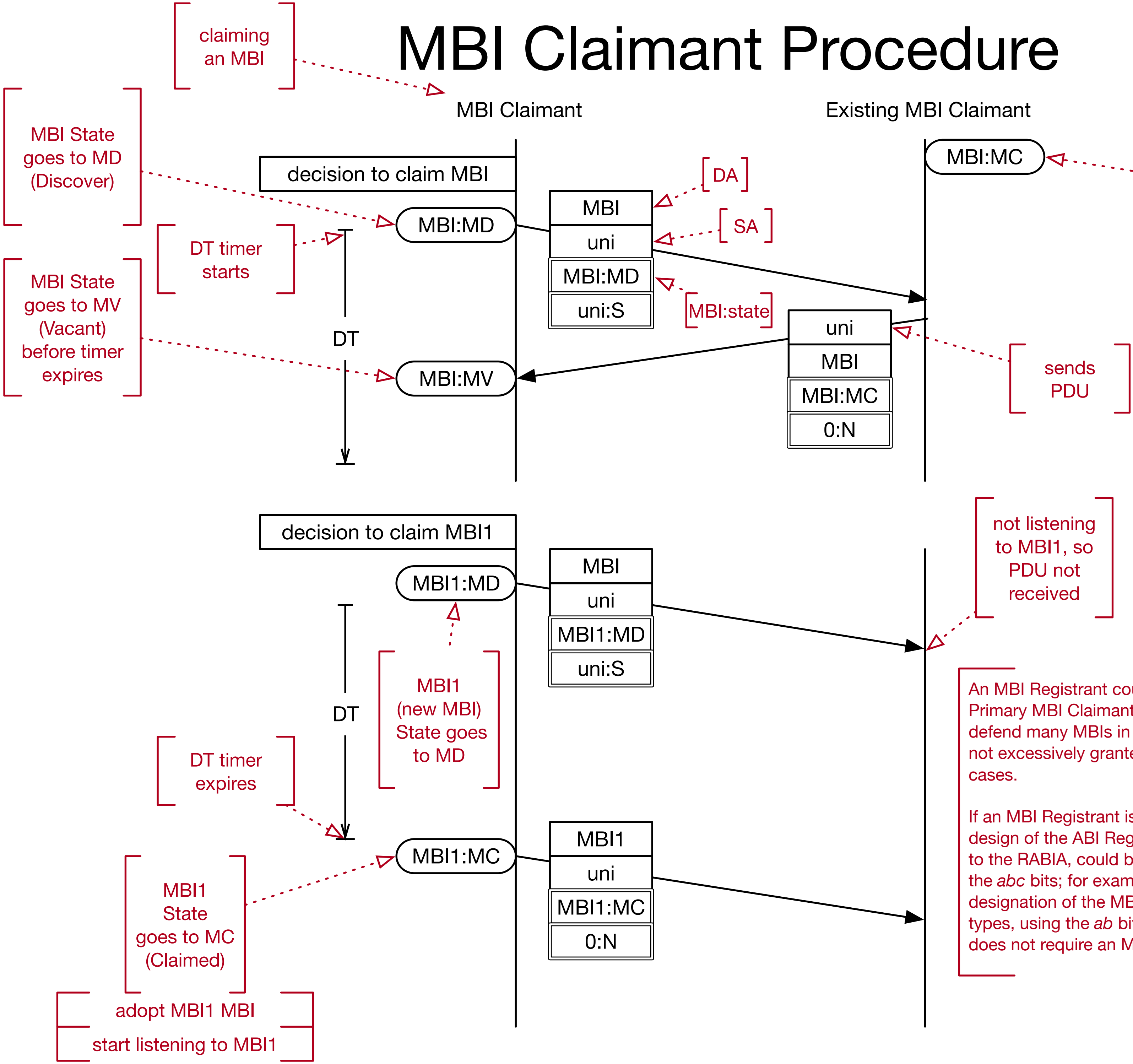
- RABIA of an active RABIA State Machine
- MBI of an active MBI State Machine
- any CABI

and MAAP multicast address for AVTPDUs

RABI Registrar: RABIA State Transition Table

	State (RABIA)			
Event	**VACANT (V)**	**OFFERED (I)**	**REGISTERED (R)**	**EXPIRED (E)**
InviteD(CABI,RABIA,da)!	Ingress(RABIA,pass) sBARC(CABI:C,RABIA:O){da,RABIA} Start Offer_Timer **OFFERED**			Ingress(RABIA,pass) sBARC(CABI:C,RABIA:O){da,RABIA} Start Offer_Timer **OFFERED**
InviteT(S1,RABIA,da)!	Ingress(RABIA,pass) sBARC(0:S1,RABIA:O){da,RABIA} Start Offer_Timer **OFFERED**			Ingress(RABIA,pass) sBARC(0:S1,RABIA:O){da,RABIA} Start Offer_Timer **OFFERED**
BARC(R,sa,token)!		R_sa==sa R_token==token Start Register_Timer(R) sBARC(RABIA:R, R_sa:S, token){R_sa,RABIA} **REGISTERED**	if sa= R_sa and token= R_token then Start Register_Timer(R)	
Offer_Timer!		Ingress(RABIA,filter) Start Expire_Timer **EXPIRED**		
Register_Timer!			sBARC(RABIA:V, R_sa:S,token){R_sa,RABIA} Ingress(RABIA,filter) Start Expire_Timer **EXPIRED**	
BARC(V,sa,da)!			if sa= R_sa and token= R_token then Ingress(sa,filter) Start Expire_Timer **EXPIRED**	
Expire_Timer!			sBARC(RABIA:V, R_sa:S,token){R_sa,RABIA} Ingress(RABIA,filter) Start Expire_Timer **EXPIRED**	**VACANT**

MBI Claimant Procedure



An MBI Registrant could be specified. Alternatively, a Primary MBI Claimant could be configured to hold and defend many MBIs in reserve, ensuring that MBIs are not excessively granted. This would suffice in many cases.

If an MBI Registrant is to be specified, following the design of the ABI Registrant, an MBIA identifier, parallel to the RABIA, could be specified by taking a bit from the *abc* bits; for example, turning the *c* bit into a designation of the MBIA and limiting the RABI to four types, using the *ab* bits. However, an MBI Registrant does not require an MBIA identifier.

MBI Claimant: MBI State Transition Table

		State (MBI)	
Event	**VACANT (MV)**	**DISCOVERY (MD)**	**CLAIMED (MC)**
Try(sa)!	sBARC(MBI:MD,sa:S){MBI,sa} Start MDiscoverTimer **DISCOVERY**		
MDiscoverTimer!		ingress(MBI,pass) Outcome(MC) sBARC(MBI:MC,0:N){MBI} Start Renew_Timer **CLAIMED**	
BARC(MC)!		Outcome(MV) **VACANT**	sBARC(MBI:MC,0:N){MBI} FYI(Alert)
BARC(MD,SA)!		Outcome(MV) **VACANT**	da=SA sBARC(MBI:MC,0:N){da}
Delete()!		**VACANT**	sBARC(MBI:MC,0:N){MBI} ingress(MBI,filter) **VACANT**
Renew_Timer! or Renew()!			sBARC(MBI:MC,0:N){MBI} FYI(Renewed) Start Renew_Timer

VLANs

- All state machines are specified per VLAN.

- All address assignments are specific to the VLAN in which the state machine operates.

- All addresses adopted are specific to the VLAN under which the assignment was completed.

- Usage of any address is limited to the VLAN under which it was obtained.

- Any address assigned within the context of a VLAN shall not be reassigned except within the context of the VLAN in which it was assigned.

- Due to the possibility that the same unicast address may be assigned in different VLANs, Independent VLAN Learning (IVL) is required in bridges, per IEEE Std 802.1Q Annex F (F.1.2).
  - This requirement could be relaxed when assigned unicast addresses are declared via MMRP.

Summary

- Claimants operate with or without Registrars.

- Multiple registrars are supported, operating with disjoint multi-blocks.

- The block discretization provides:
  - a vast set of addresses to a LAN
    - though operating entirely within 1/16 of the SAI quadrant of local address space
  - a large set of temporary unicast addresses
  - operational efficiency and simplicity
  - both unicast and multicast addresses (1/2 or 2/3 unicast) to Claimant
    - including one unicast and multicast subblock with the same range, except for the M bit
      - could be exploited
    - devices needing both unicast and multicast addresses need make only one claim

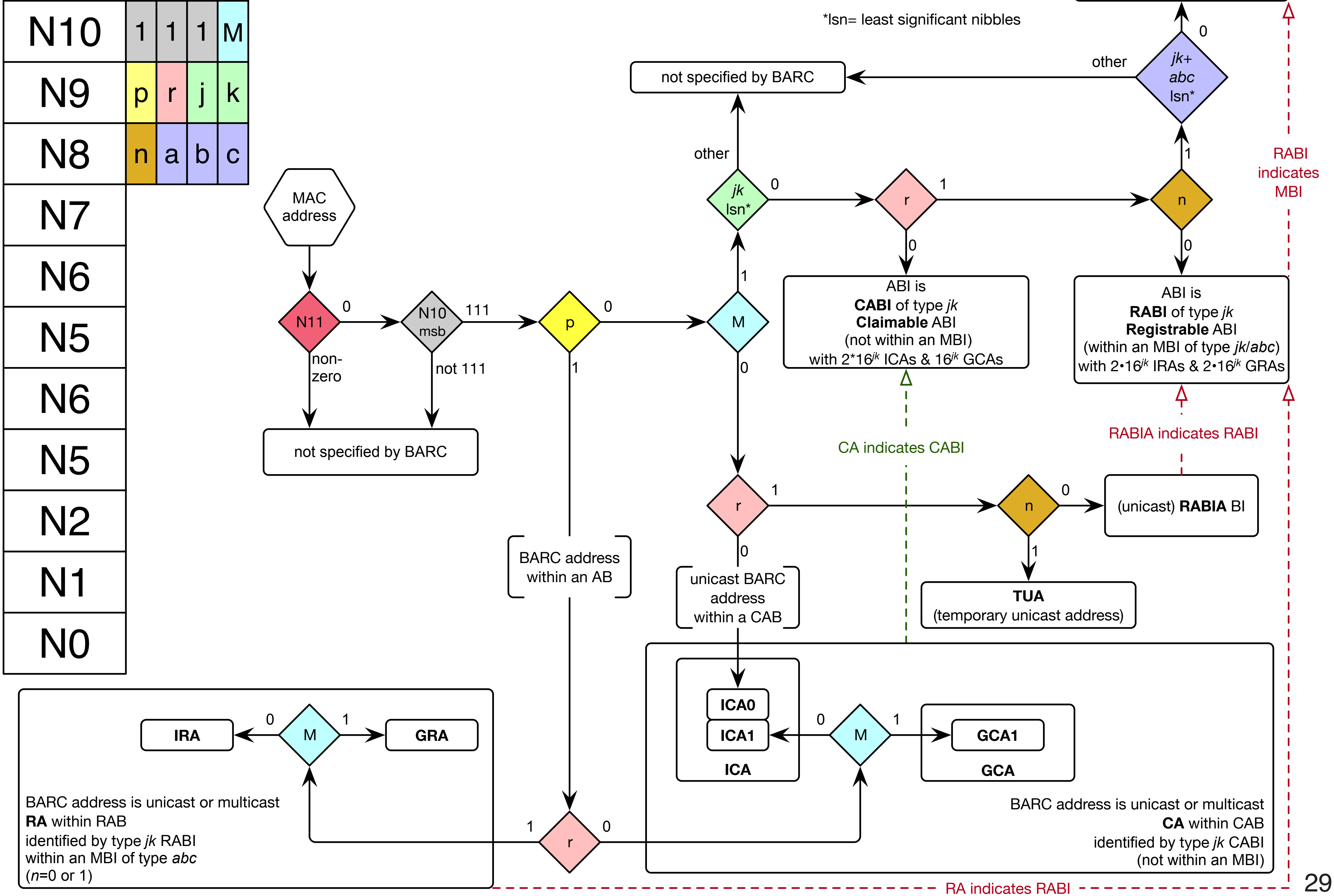
- Could integrate with MMRP to limit propagation and eliminate learning of unicast AB content.
  - MMRP needs to efficiently handle address ranges
  - BARP could be specified as alternative MRP application

Appendix 1

- additional details on BARC addresses and identifiers

MAC Address Parsing

N11	0			
N10	1	1	1	M
N9	p	r	j	k
N8	n	a	b	c
N7				
N6				
N5				
N6				
N5				
N2				
N1				
N0				



CABI and CA

MBI, RABI, RA, RABIA

nibble	CABI	CA1	CA0	MBI	RABI	RA	RABIA
N11	0	0	0	0	0	0	0
N10	1 1 1 1	1 1 1 *	1 1 1 0	1 1 1 1	1 1 1 1	1 1 1 *	1 1 1 0
N9	0 0 j k	1 0 j k	0 0 j k	0 1 j k	0 1 j k	1 1 j k	0 1 j k
N8	X8	X8	X8	1 a b c	0 a b c	* a b c	0 a b c
N7	X7	X7	X7	X7	X7	X7	X7
N6	X6	X6	X6	X6	X6	X6	X6
N5	X5	X5	X5	X5	X5	X5	X5
N4	X4	X4	X4	X4/0	X4/#	X4/#	X4/#
N3	X3	X3	X3	X3/0	X3/#	X3/#	X3/#
N2	X2/0	X2/*	X2/*	X2/0	X2/0	X2/*	X2/*
N1	X1/0	X1/*	X1/*	X1/0	X1/0	X1/*	X1/*
N0	X0/0	X0/*	X0/*	X0/0	X0/0	X0/*	X0/*

MBI identifies its RABIs

abc nibbles for MBI type *jk/abc*[+]

jk nibbles for ABI type *jk*

[+]*abc*=8 when *a*=*b*=*c*=0

** indicates all possible values*

 # indicates all possible values

Four CABI Types

CABI Type 0			CABI Type 1			CABI Type 2			CABI Type 3		
CABI	CA1	CA0	CABI	CA1	CA0	CABI	CA1	CA0	CABI	CA1	CA0
0	0	0	0	0	0	0	0	0	0	0	0
1 1 1 1	1 1 1 *	1 1 1 0	1 1 1 1	1 1 1 *	1 1 1 0	1 1 1 1	1 1 1 *	1 1 1 0	1 1 1 1	1 1 1 *	1 1 1 0
0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 1	1 0 0 1	0 0 0 1	0 0 1 0	1 0 1 0	0 0 1 0	0 0 1 1	1 0 1 1	0 0 1 1
X8	X8	X8	X8	X8	X8	X8	X8	X8	X8	X8	X8
X7	X7	X7	X7	X7	X7	X7	X7	X7	X7	X7	X7
X6	X6	X6	X6	X6	X6	X6	X6	X6	X6	X6	X6
X5	X5	X5	X5	X5	X5	X5	X5	X5	X5	X5	X5
X4	X4	X4	X4	X4	X4	X4	X4	X4	X4	X4	X4
X3	X3	X3	X3	X3	X3	X3	X3	X3	X3	X3	X3
X2	X2	X2	X2	X2	X2	X2	X2	X2	0	*	*
X1	X1	X1	X1	X1	X1	0	*	*	0	*	*
X0	X0	X0	0	*	*	0	*	*	0	*	*

- 3 contiguous subblocks per CABI (CA0 and CA1 unicast, CA1 multicast)

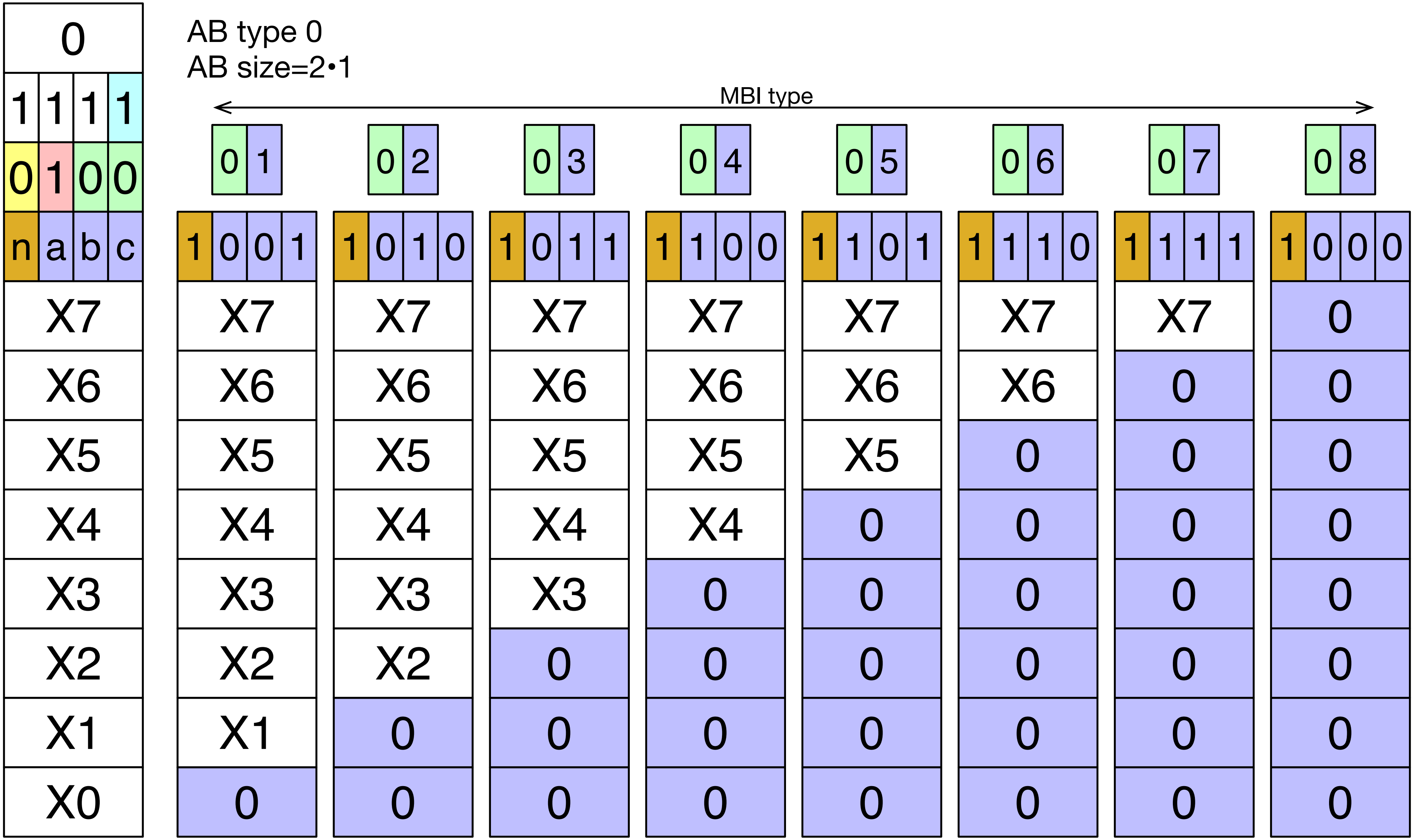
- 6.9E10 Type 0 CABIs
- 1 address per subblock
- 3 addresses/CABI

- 4.3E9 Type 1 CABIs
- 16 addresses per subblock
- 48 addresses/CABI

- 2.7E8 Type 2 CABIs
- 256 addresses per subblock
- 768 addresses/CABI

- 1.6E7 Type 3 CABIs
- 4096 addresses per subblock
- 12288 addresses/CABI

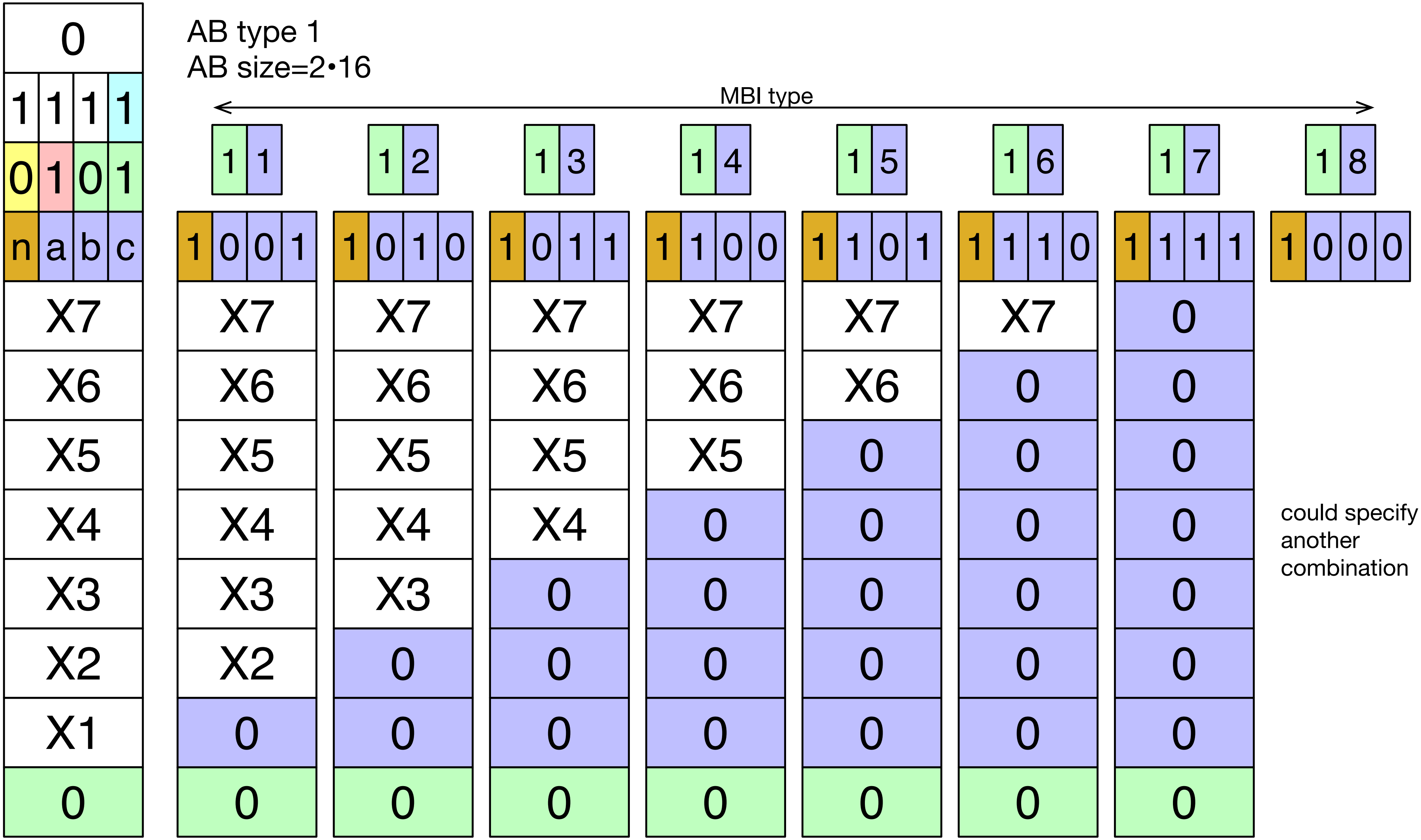
Multi-Blocks and Multi-Block Identifiers (MBIs)



MBI size	16	256	4096	65k	1M	17M	268M	4.3E9
MBI count	268M	17M	1M	65k	4096	256	16	1
AB count	4.3E9	4.3E9	4.3E9	4.3E9	4.3E9	4.3E9	4.3E9	4.3E9
addresses*	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9

*total for each case: unicast and multicast

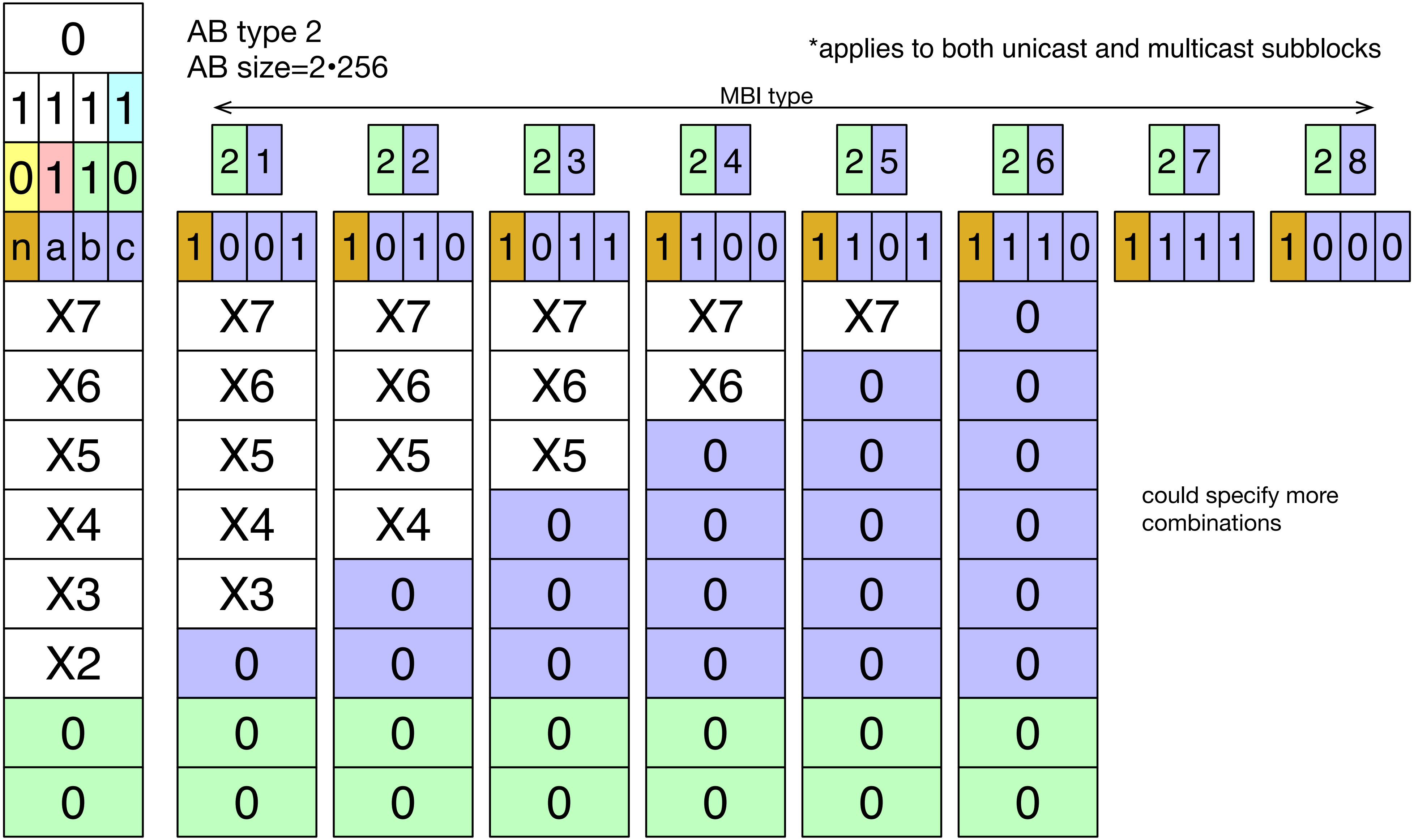
Multi-Blocks and Multi-Block Identifiers (MBIs)



MBI size	16	256	4096	65k	1M	17M	268M
MBI count	17M	1M	65k	4096	256	16	1
AB count	268M	268M	268M	268M	268M	268M	268M
addresses*	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9

*total for each case: unicast and multicast

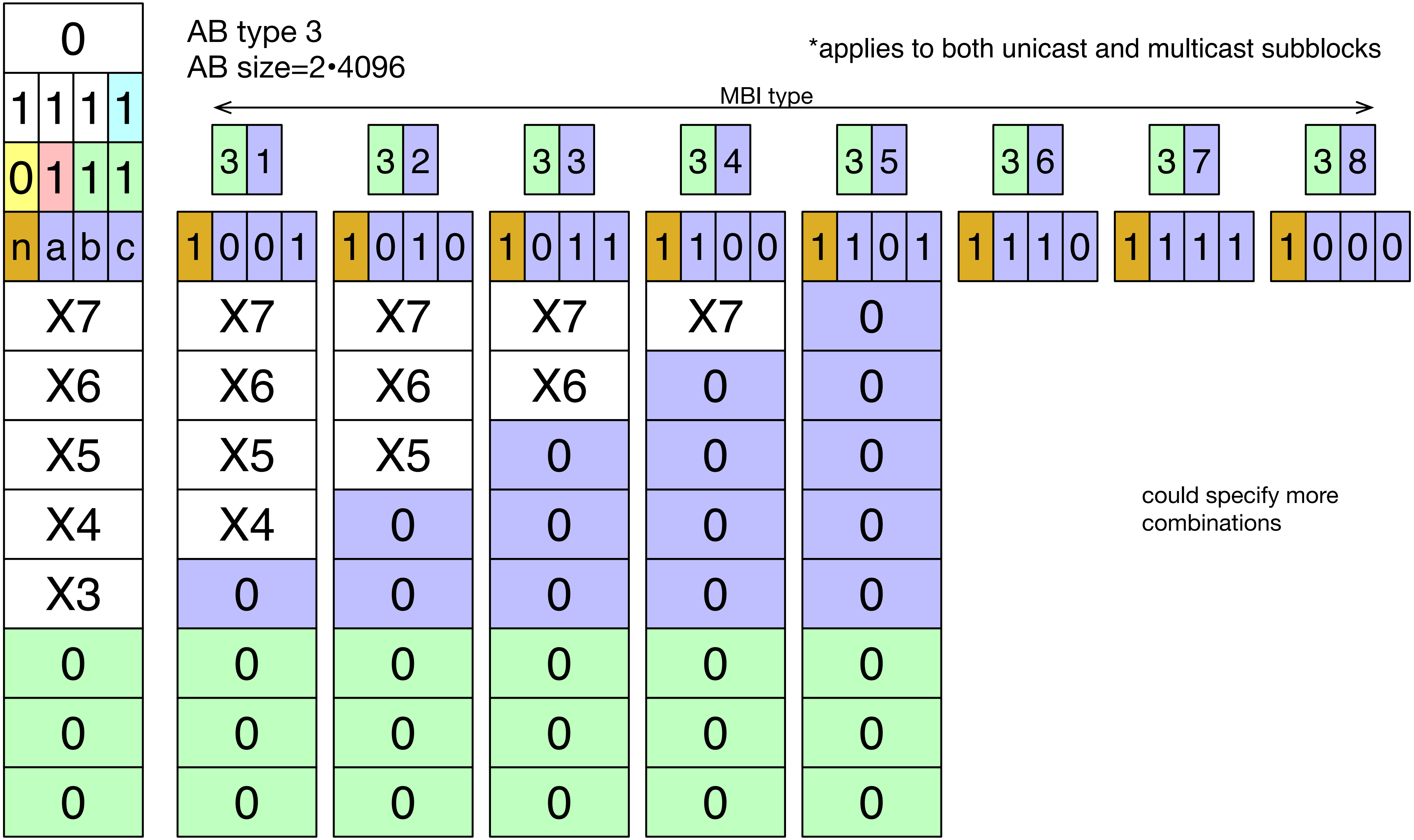
Multi-Blocks and Multi-Block Identifiers (MBIs)



MBI size	16	256	4096	65k	1M	17M
MBI count	1M	65k	4096	256	16	1
AB count	17M	17M	17M	17M	17M	17M
addresses*	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9

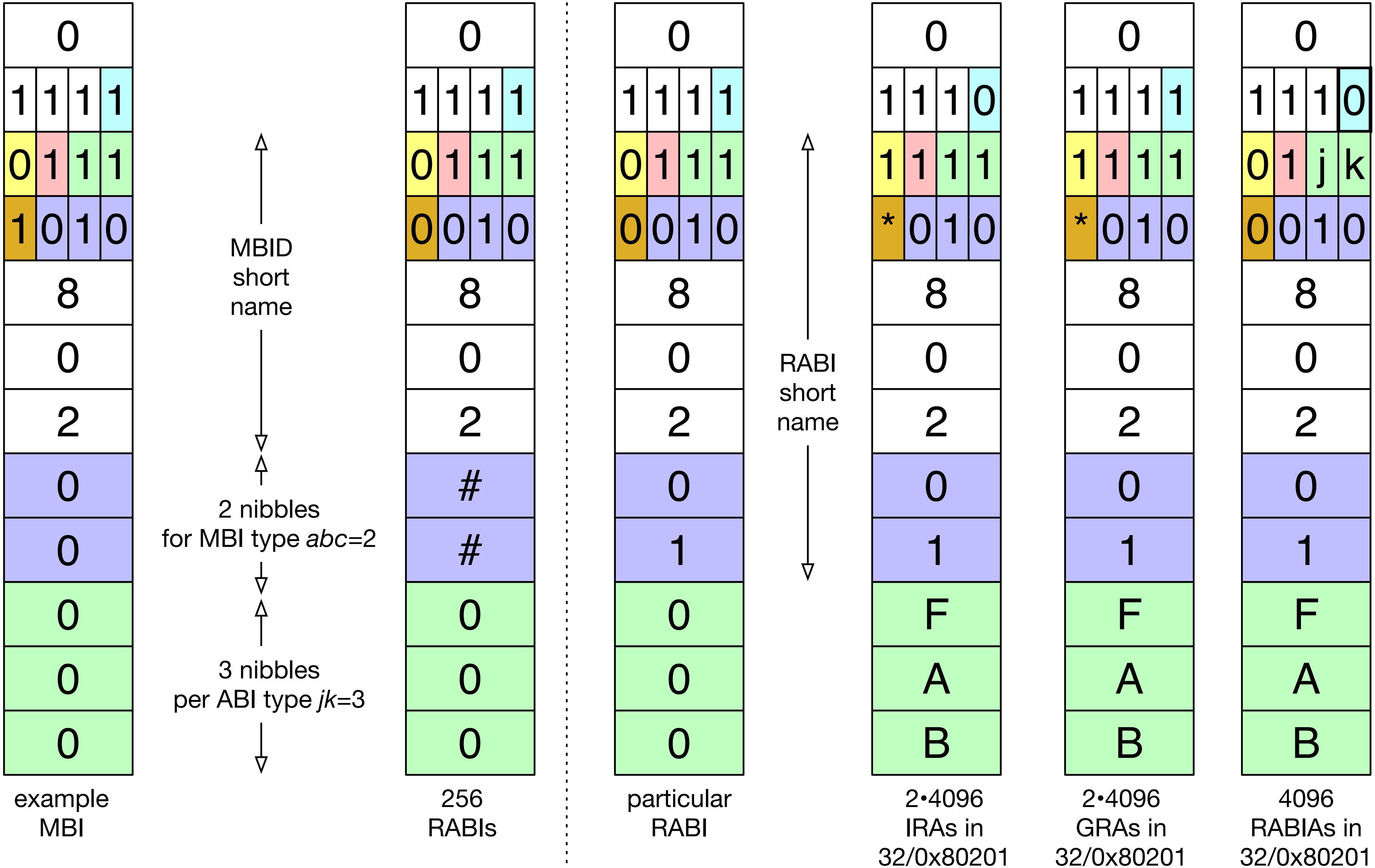
*total for each case: unicast and multicast

Multi-Blocks and Multi-Block Identifiers (MBIs)



MBI size	16	256	4096	65k	1M
MBI count	65k	4096	256	16	1
AB count	1M	1M	1M	1M	1M
addresses*	8.6E9	8.6E9	8.6E9	8.6E9	8.6E9

MBI Example RABI Example RA, RABIA Example



MBI short name: 32/0x802 (32=MBI Type)

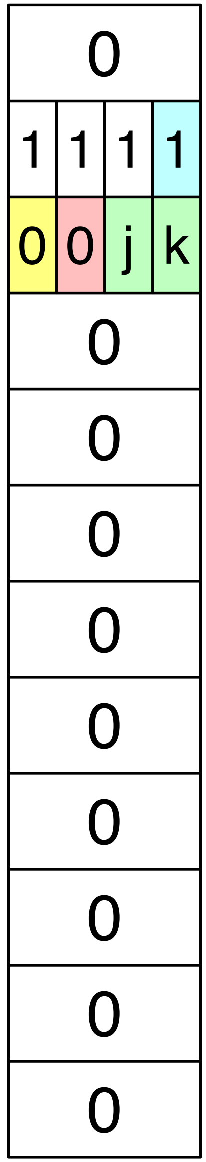
RABI short name: 32/0x80201 (32=MBI Type)

 # indicates all possible values

** indicates all possible values*

null CABI: identifies no AB

- Could be used when initiating discovery to expected Registrar.
- Conveys to Registrar only the size of the requested AB.
- No other Claimant listens to this address.



Non-AB Temporary Unicast Address (TUA)

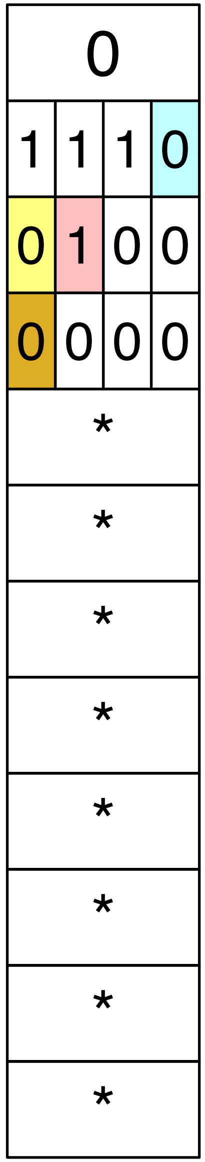
- For temporary use

- device without a source address selects a random non-AB temporary unicast address for initial discovery <u>only</u>
  – protocol then assigns at least one persistent unicast address

- simultaneous duplicate temporary addresses may lead to message loss in some circumstances
  – network learns route to source as initial message crosses the network
  – before response is returned, another initial message with duplicate source address crosses the path and rewrites the route
  – unlikely to be disastrous
  – loss of initial message will be corrected eventually

- nevertheless, need to consider the likelihood of duplication

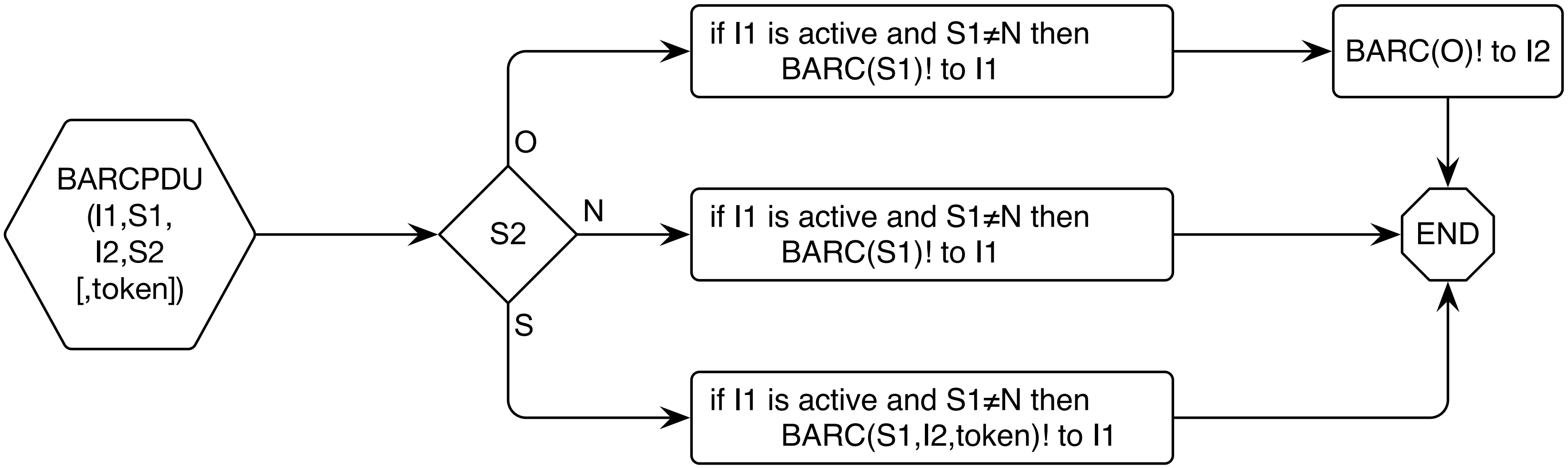
- Temporary address range includes 8 full nibbles of 16 values each (0–F)
  – 16^9 = 68,719,476,736 (= N) temporary addresses in the pool
  – chance of no duplicates with k randomly selected addresses is approximated $\exp(-k*(k-1)/(2*N))$
  – with k=1000 devices <u>simultaneously</u> using a temporary address, chance of no duplicates is ~0.99988
  – address conflicts are rare, usually not harmful, and recoverable
  – can add first 2 bits of the N9 nibble and first 3 bits of the N8 nibble to the pool chance of no duplicates is then ~0.999996 (k=1000)



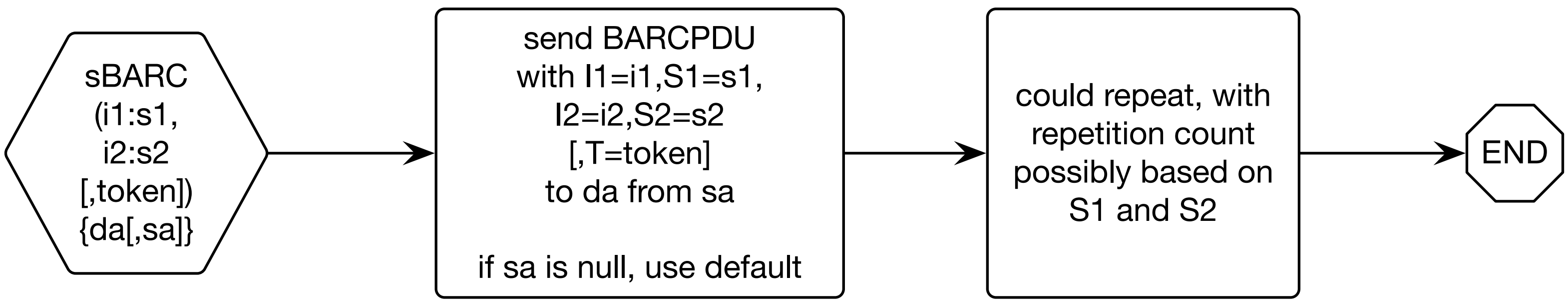
Appendix 2

- additional procedural details

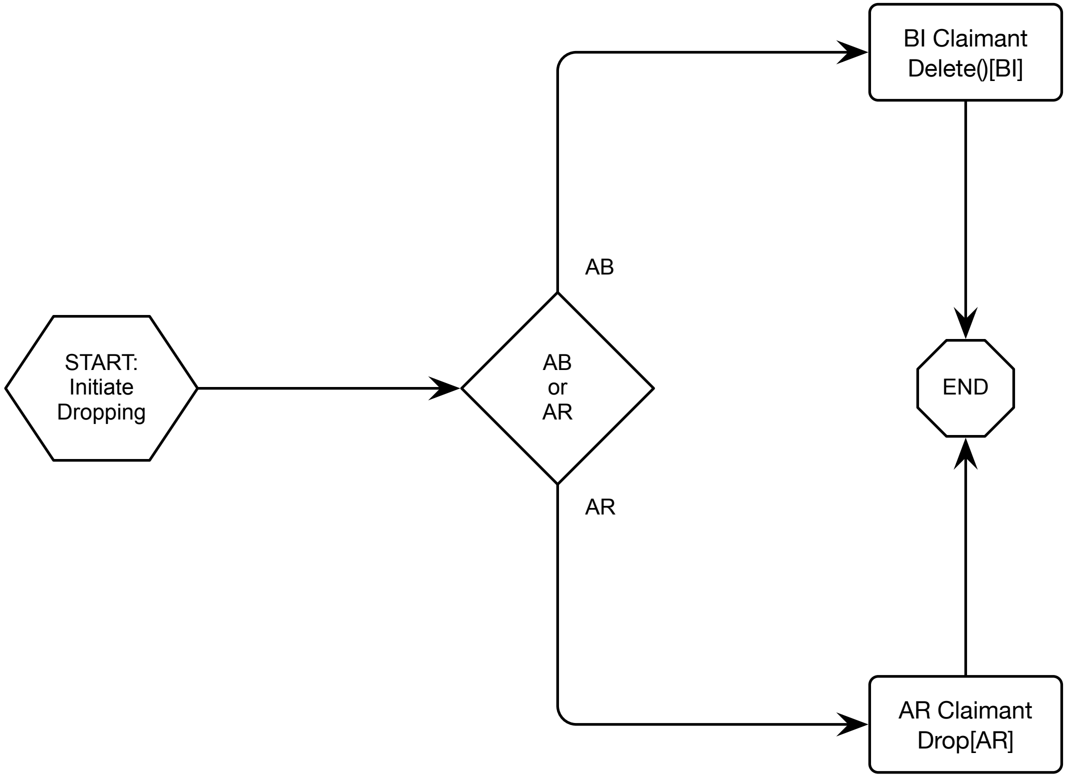
ARC Claimant: BARCPDU Processor – ingress



ARC Claimant: BARCPDU Processor – egress



ARC Claimant Application Process: Drop Claim



BARC Address Propagation with MMRP

The ARC Claimant Application Process includes “declare with MMRP”. This entails declaring, to MMRP (when available), MMRP attributes, using an MMRPDU per IEEE Std 802.1Q § 10.12.1.6:

- The multicast address represented by the ABI
  - FirstValue field = ABI/NumberOfValues=1
- The two unicast address set subblocks indicated by the ABI (CABI or RABI)
  - FirstValue field = first ABI in unicast subblock/NumberOfValues = 16^{jk} per *jk* in nibble N9 of ABI
    - maximum NumberOfValues is with *jk*=3; 16^3=4096; MRP provides 13 bits of NumberOfValues (2^{13}=8192)

The ARC Claimant Application Process includes (“select CABI”); this selection should consider any local MMRP registration database to avoid selecting a registered CABI.

Unicast MMRP declaration can be useful because:
(1) A one-step declaration covers a contiguous range of self-assigned unicast addresses.
(2) Eliminates flooding for all the unicast addresses in the assignment.
(3) Eliminates the need for learning of each unicast address when used.
(4) Precludes erroneous re-learning of an address when a false duplicate is used elsewhere in the network.

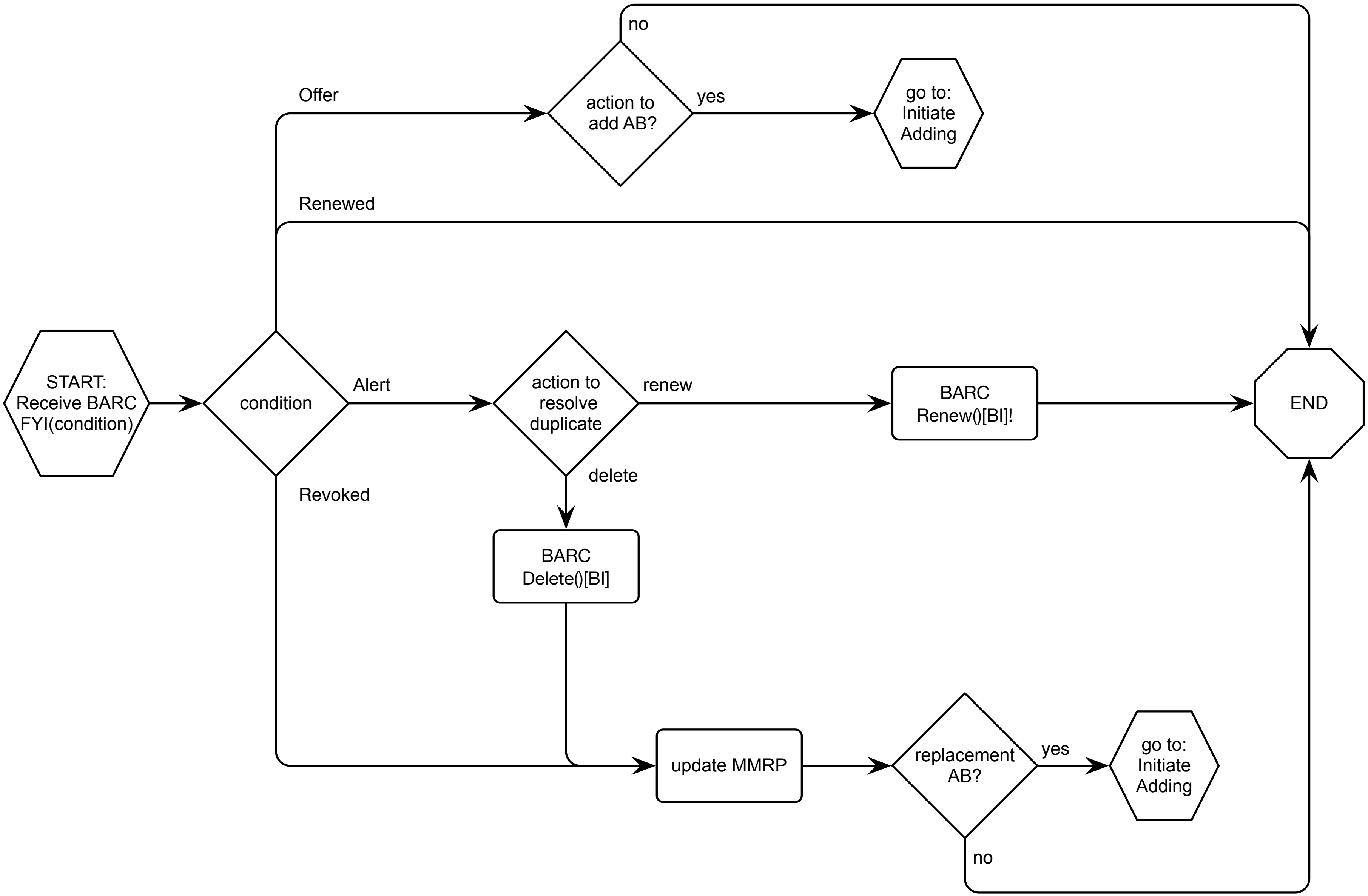
- Could be a way to control duplication.
- Security issues to study.

BARC could alternatively specify “BARP,” a new MRP application. This could entail the following changes:

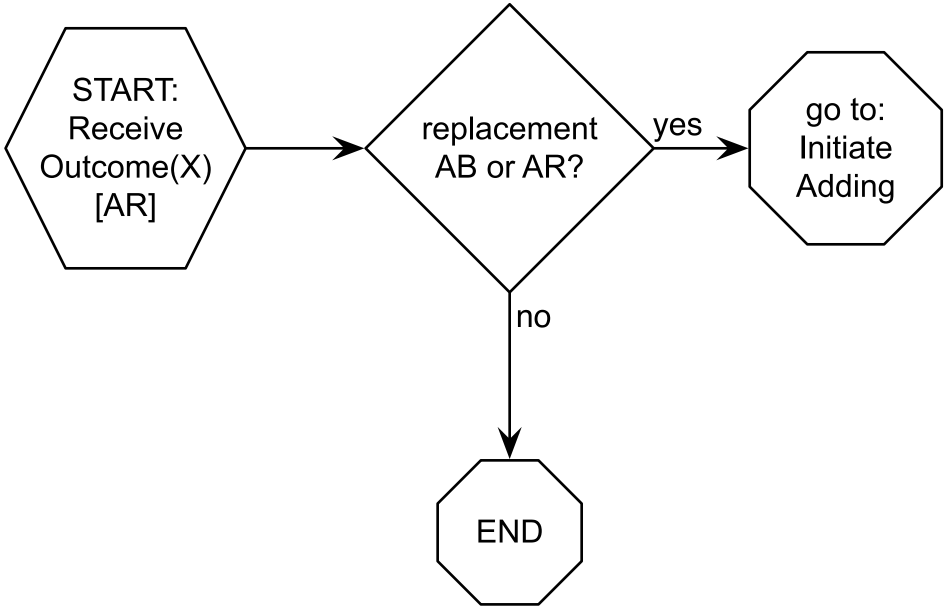
(a) the BARP application would be enabled to Join and Leave with the ABI as the declared attribute
(b) the BARP application would be specified to understand the semantics of the ABI and extract from it the indicated ABI multicast address and the indicated unicast address set, then use it to populate the FDB
(c) In the BARC BI State Machine, the ABI claim [“sBARC(ABI:C)]” might not be needed, since the a BARP declaration could convey the claim to the ABI as well as the declaration of interest in receiving at the ABI multicast address

BARP might be better suited to specification within IEEE Std 802.1Q instead of 802.1CQ.

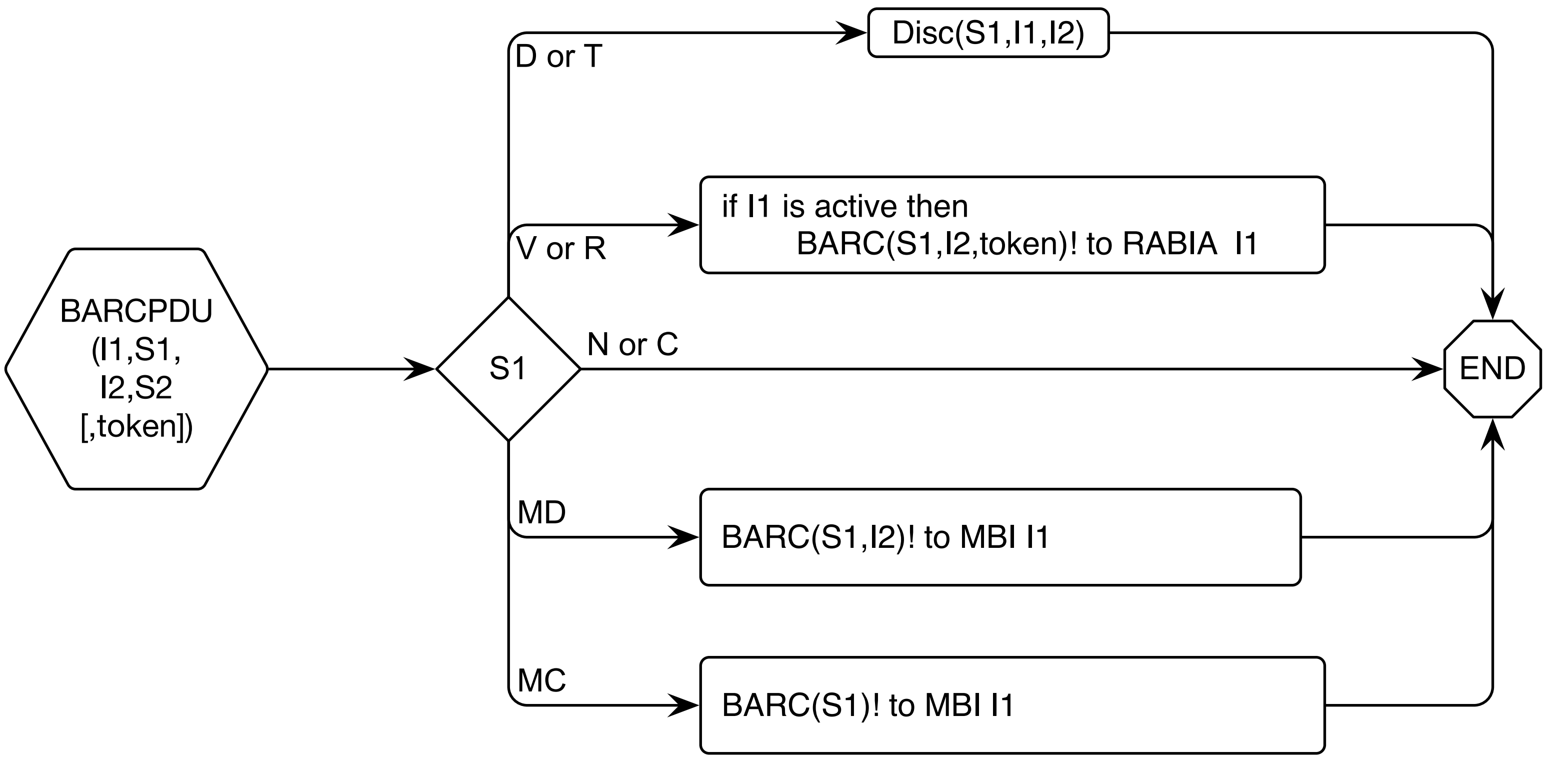
ARC Claimant Application Process: BARC Management



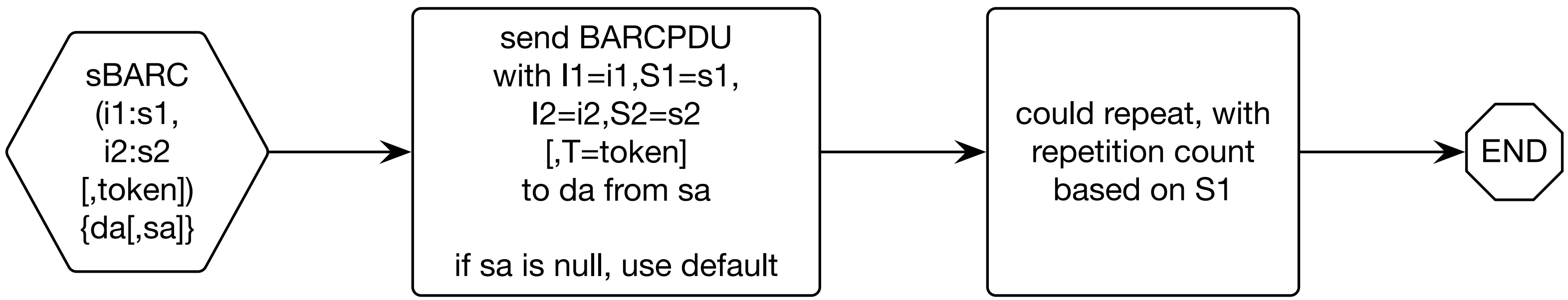
ARC Claimant Application Process: MAAP Management



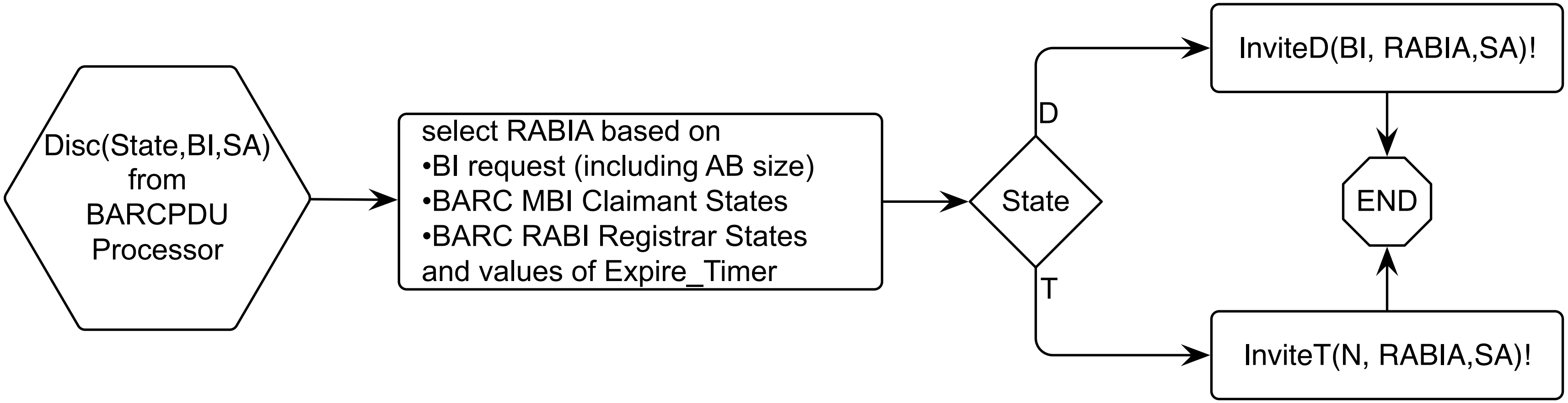
BARC Registrar: BARCPDU Processor – ingress



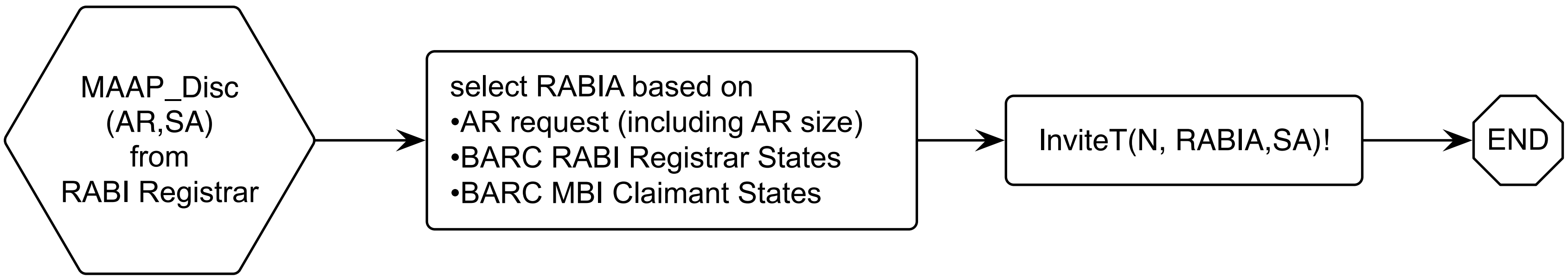
BARC Registrar: BARCPDU Processor – egress



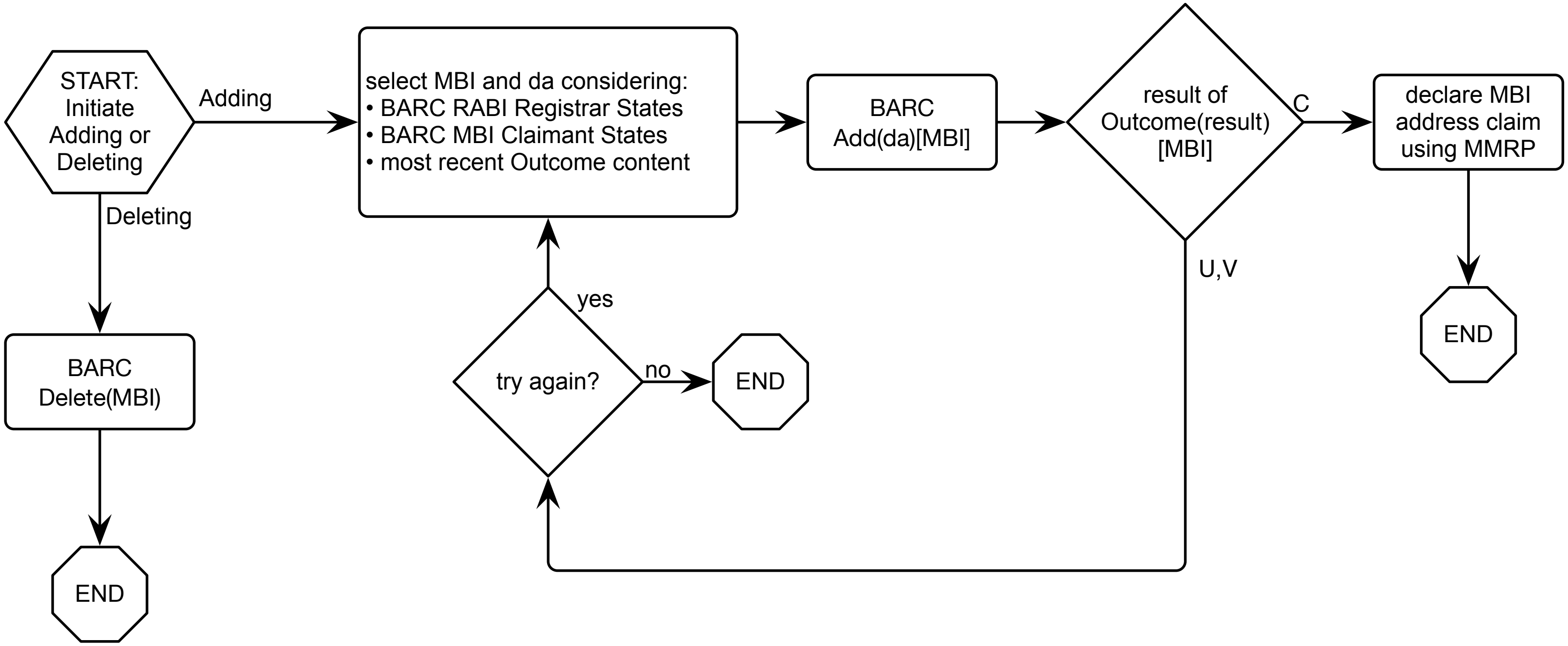
BARC Registrar Application: Disc Processing



BARC Registrar Application: MAAP_Disc Processing



BARC Registrar Application: MBI Claimant Management



MBI Claimant Application Process: BARC Management