Plexus GCC Interworking

Dillon Feng

x5268

Agenda

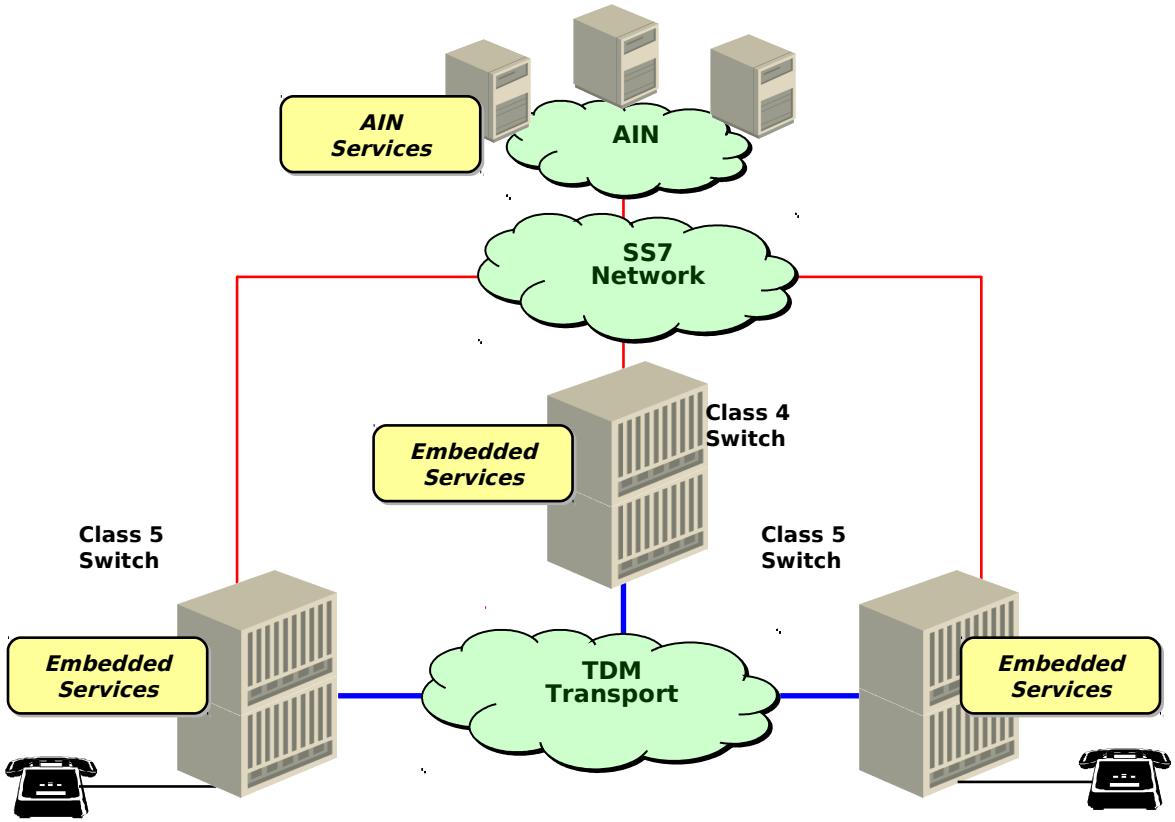
- Plexus Background
- Plexus Architecture
- GCC Introduction
- GCC State Machine
- GCC Protocol Interworking
- GCC Redundancy

Agenda

- Plexus Background
- Plexus Architecture
- GCC Introduction
- GCC State Machine
- GCC Protocol Interworking
- GCC Redundancy

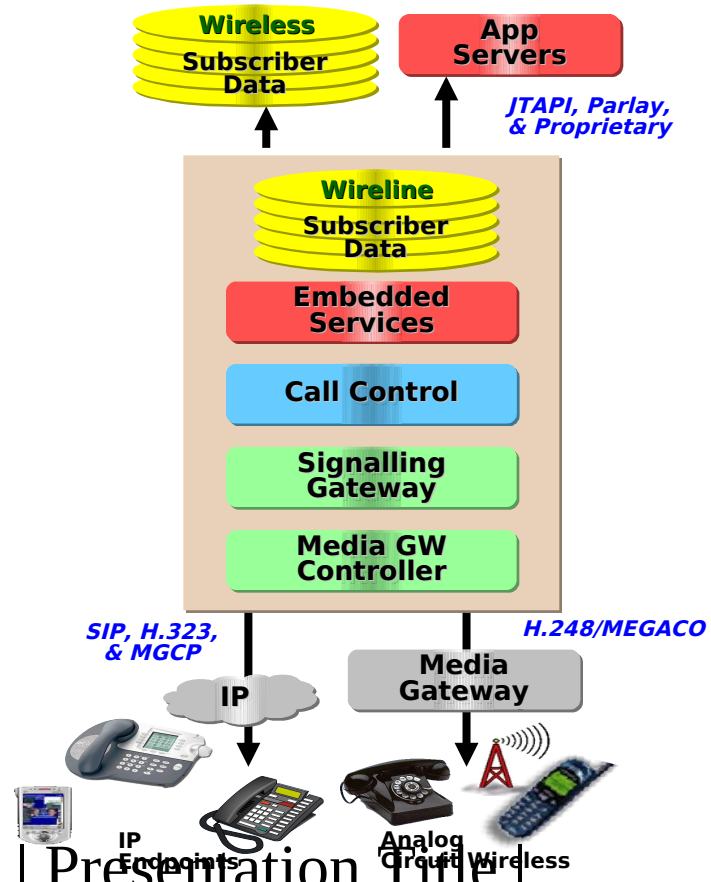
Plexus Background

- Traditional PSTN Network

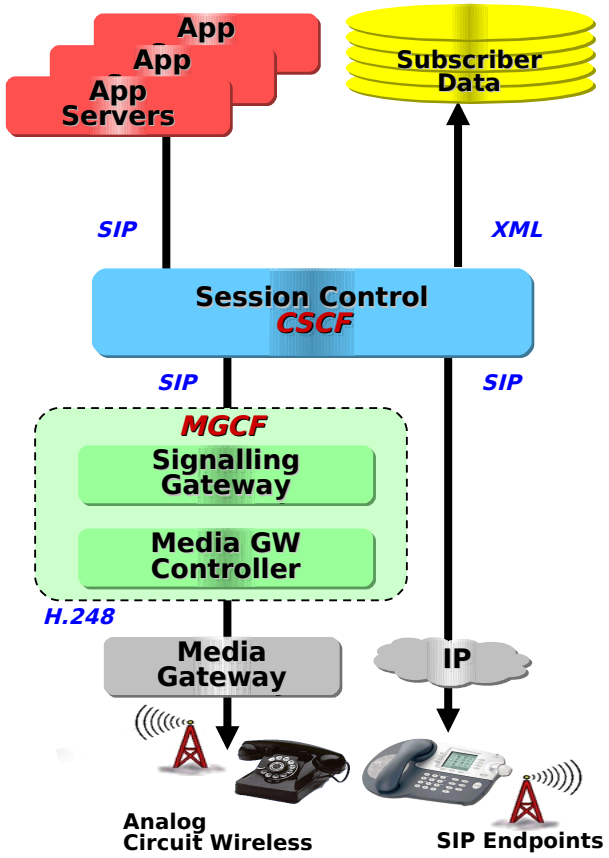


Plexus Background

- NGN Network
  - Consolidated Softswitch model

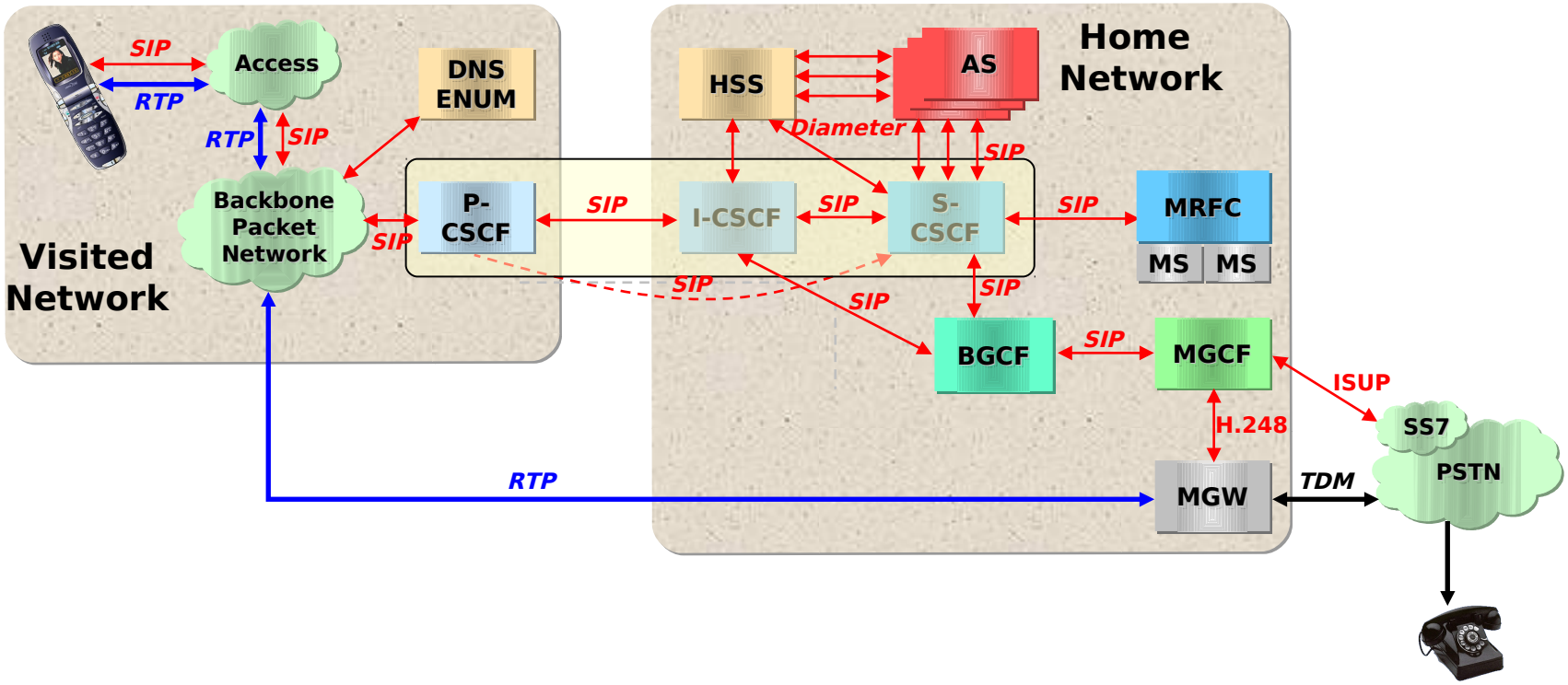


  - IMS model



Plexus Background

- IMS goal
  - Whole IP based network
  - Upgrade the exiting PSTN network

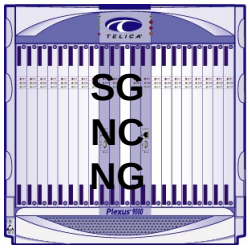


Plexus Background

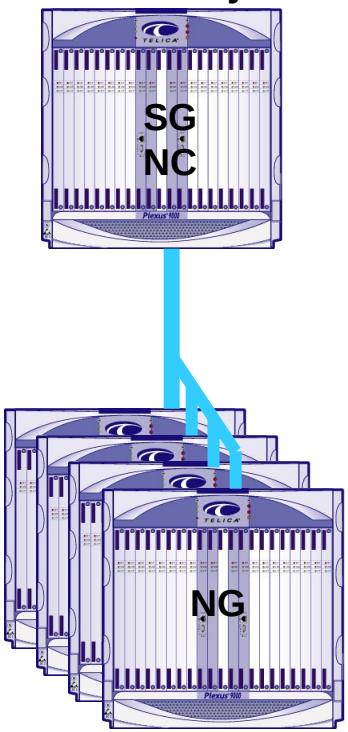
- Lucent need one bridge for IMS and Traditional network
  - Plexus is coming
    - Provide Class 5 services (3.x load)
    - Provide Class 4 services (6.x load)

Plexus Configuration

Lucent Compact Softswitch

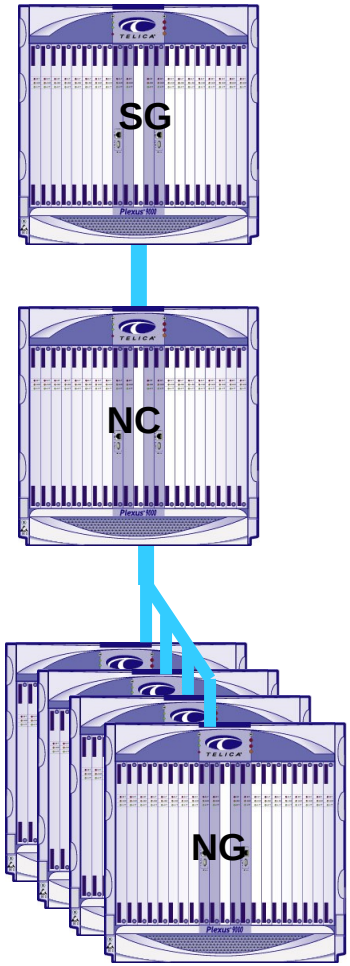


Lucent Network Controller w/Signaling Gateway



Lucent Network Gateways

Fully Distributed Configuration



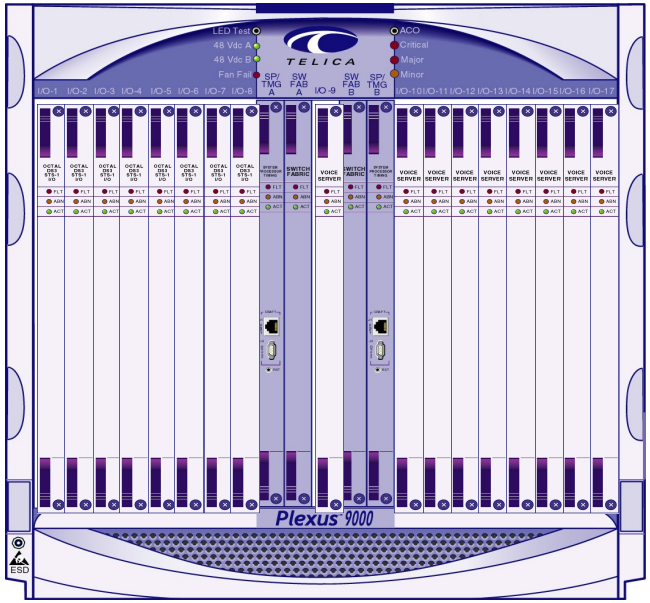
Best Flexibility
Best Scalability
Best Position for Future Growth

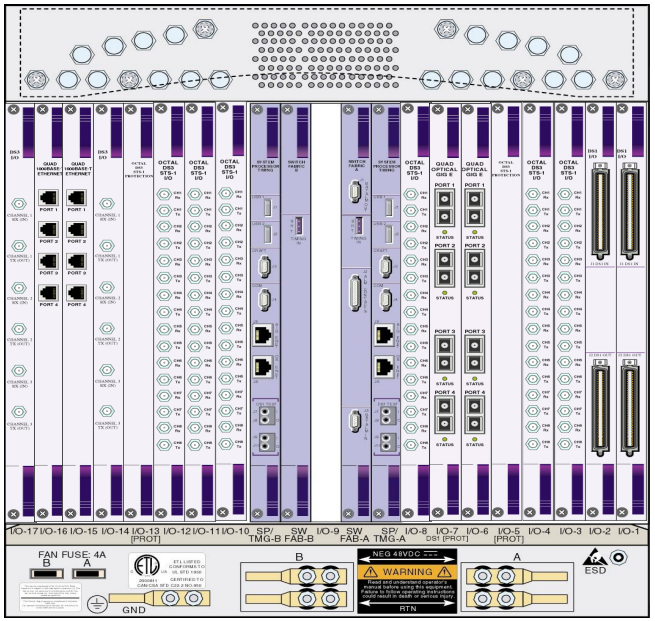
Agenda

- Plexus Background
- Plexus Architecture
- GCC Introduction
- GCC State Machine
- GCC Protocol Interworking
- GCC Redundancy

Plexus Architecture

- Hardware architecture

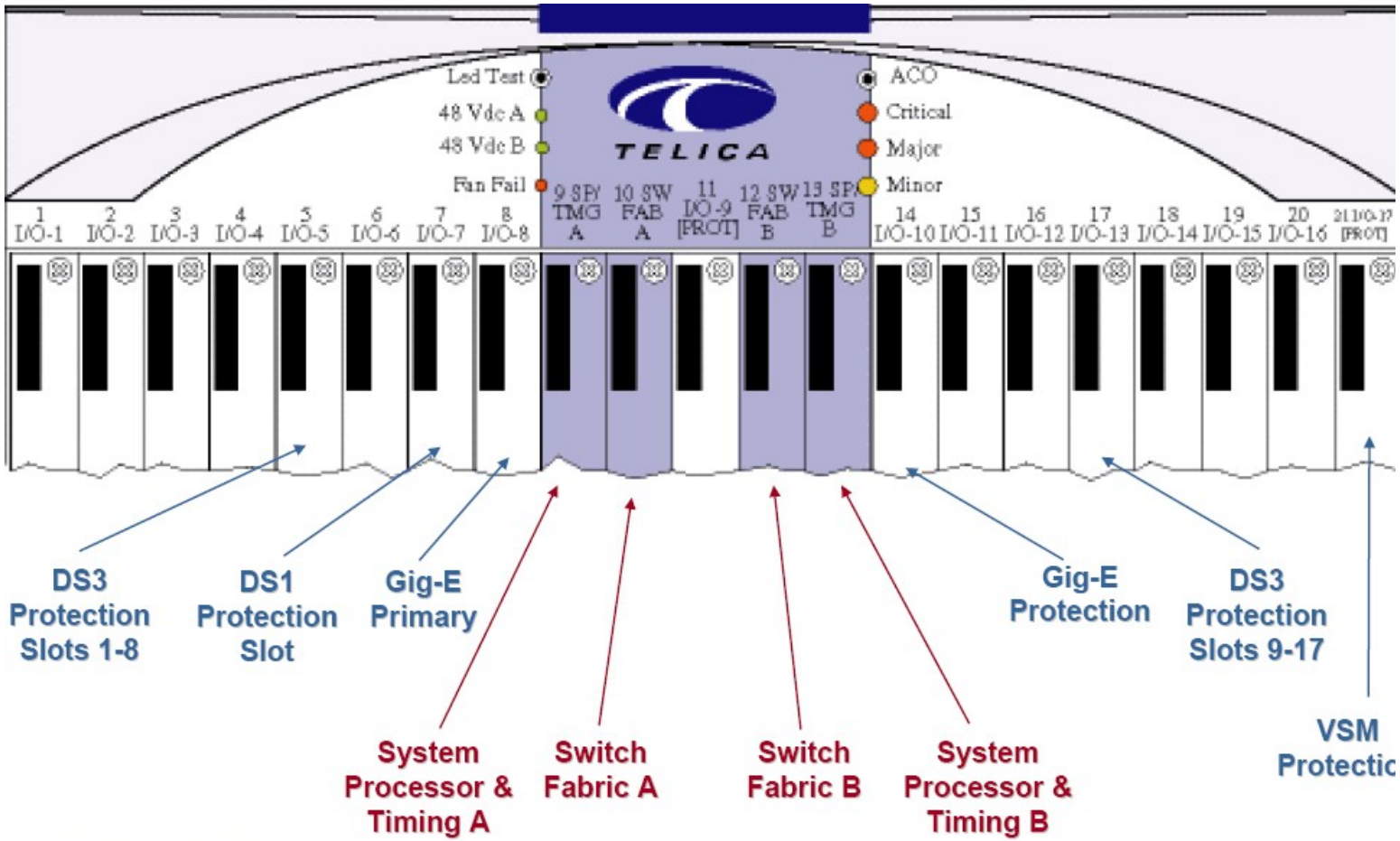




Alcatel·Lucent

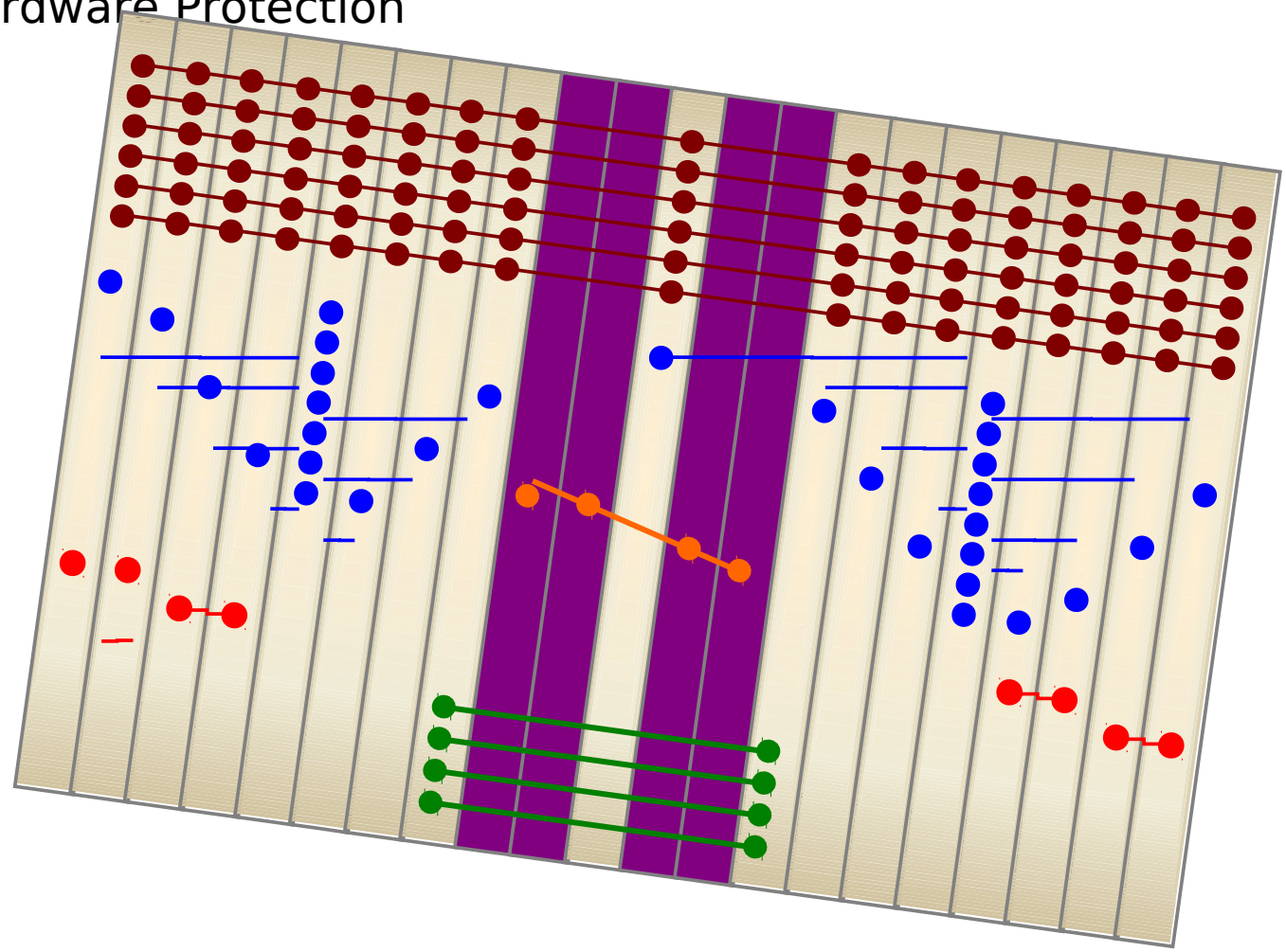
Plexus Architecture

- Hardware front view



Plexus Architecture

- Hardware Protection

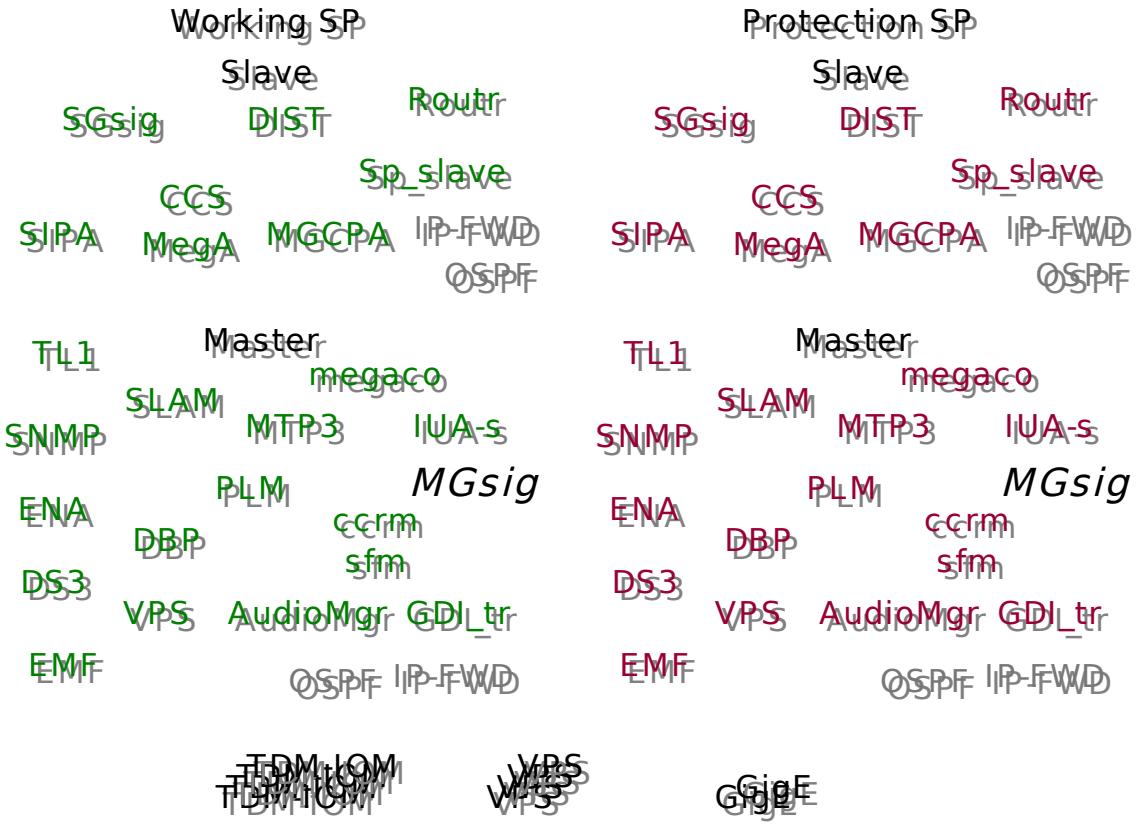


Plexus Architecture

- Software architecture
  - **MGC+SG+MG**
  - **MGC**
  - MG
  - SG
  - MGC+SG
  - SG+MG

Plexus Architecture

- Compact switch

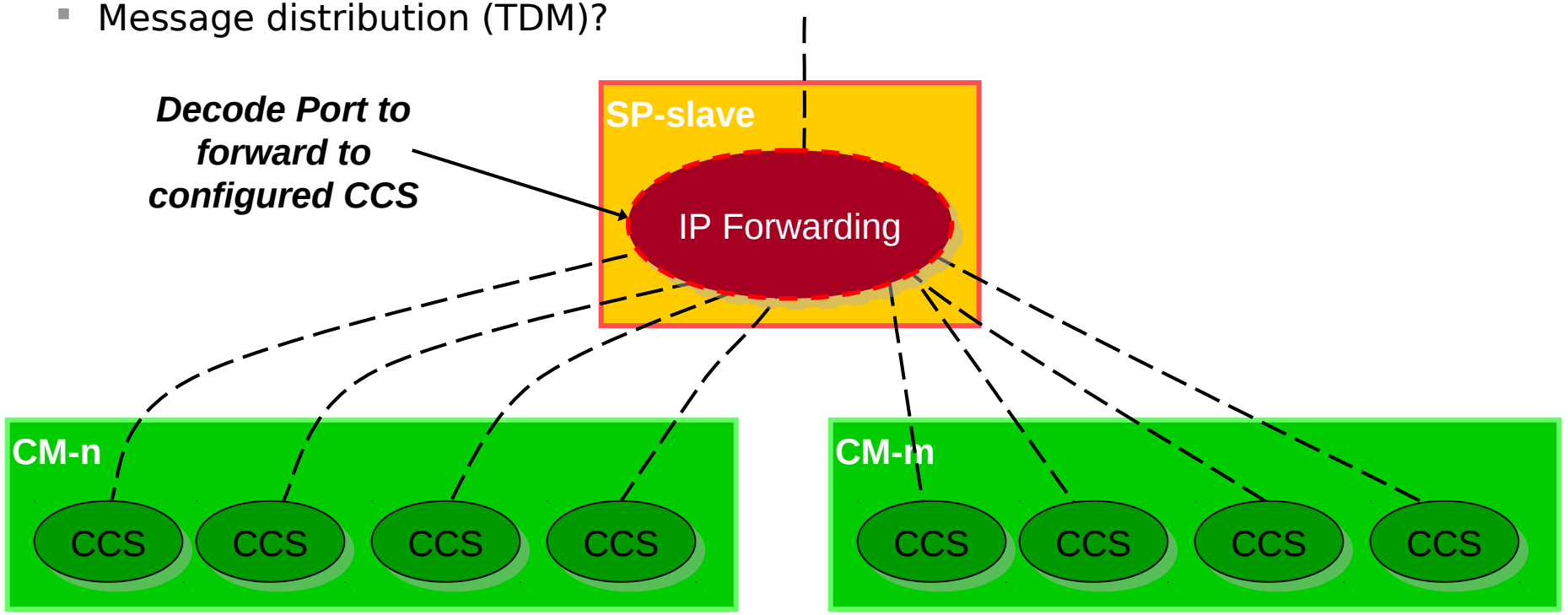


Plexus Architecture

- Fully distribute switch
  - MGC

Plexus Architecture

- Message distribution (IP)
- Message distribution (TDM)?

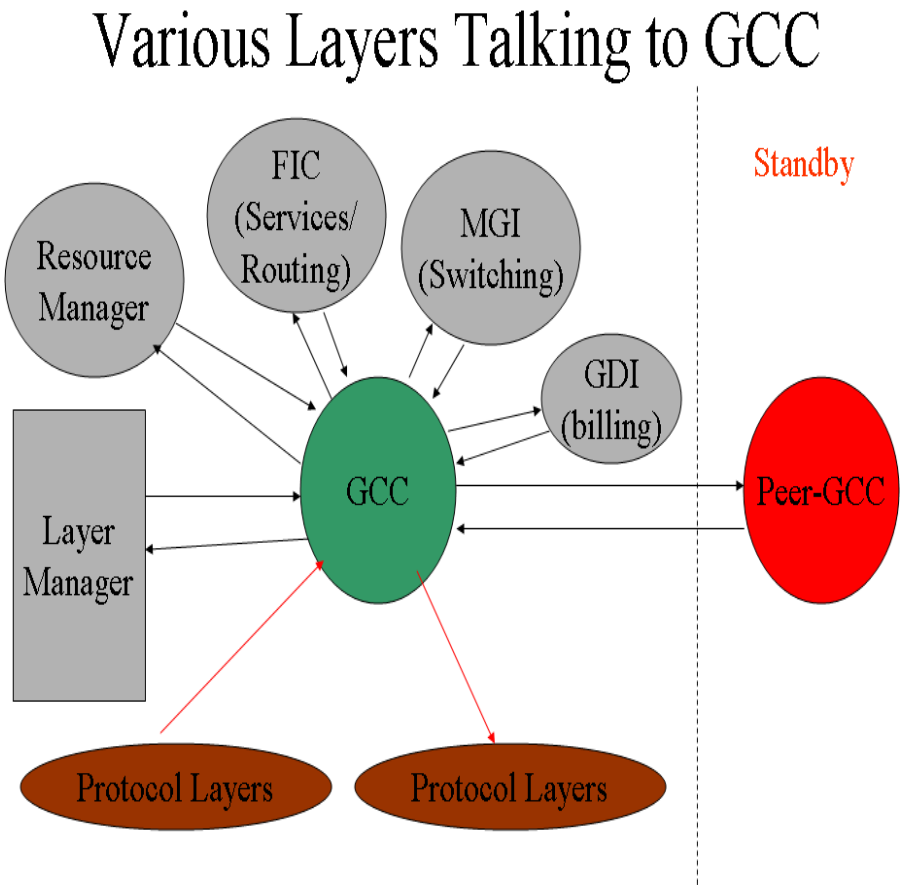


Agenda

- Plexus Background
- Plexus Architecture
- GCC Introduction
- GCC State Machine
- GCC Protocol Interworking
- GCC Redundancy

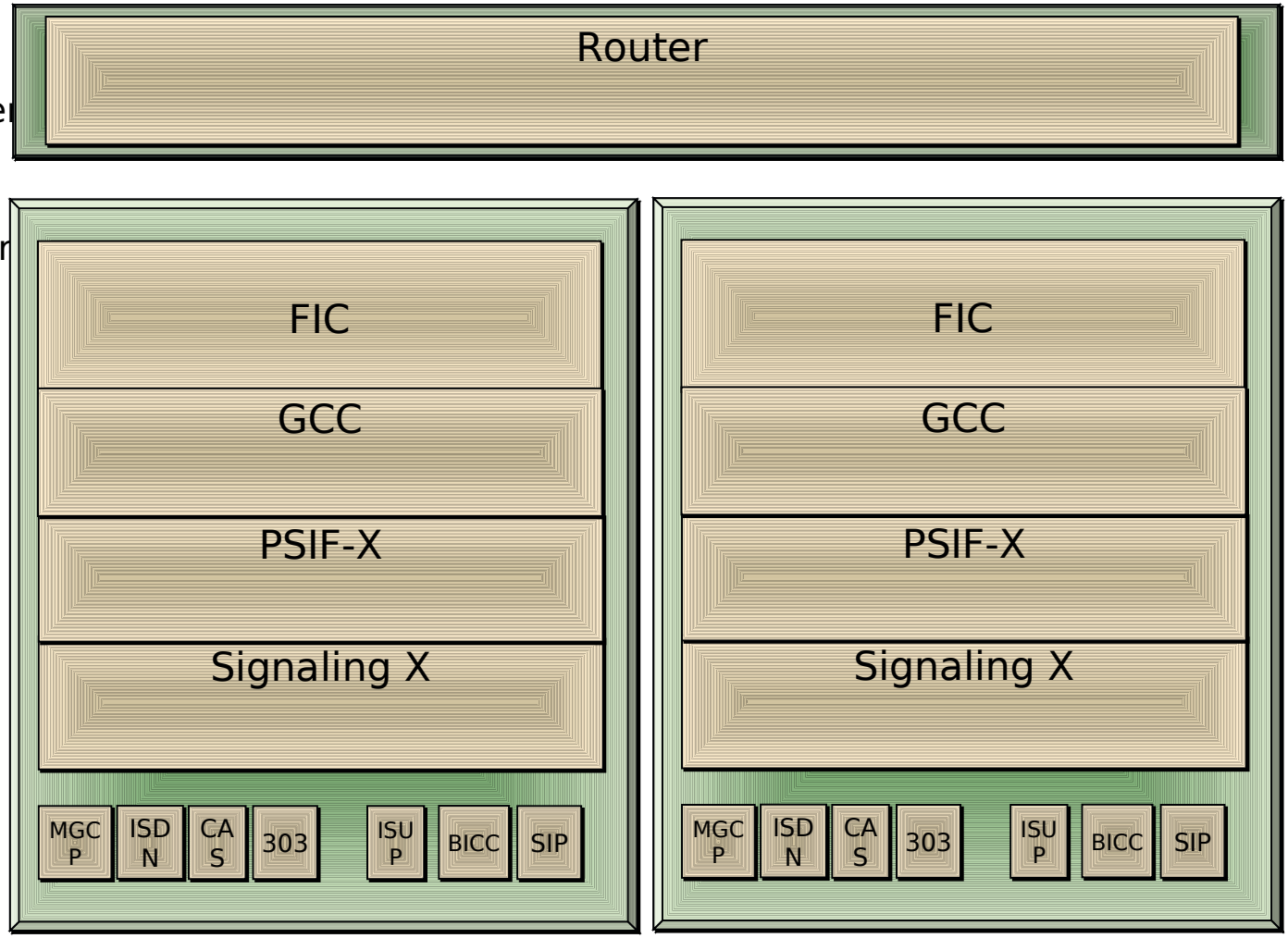
GCC Introduction

- GCC
  - Generic Call Control
  - One bridge from one protocol to the other
  - Routing call
  - redundancy
  - ……



GCC Introduction

- GCC
  - Signaling Layer
  - PSIF-X
  - Routing Server
  - FIC (service)
  - Billing Server
  - ……

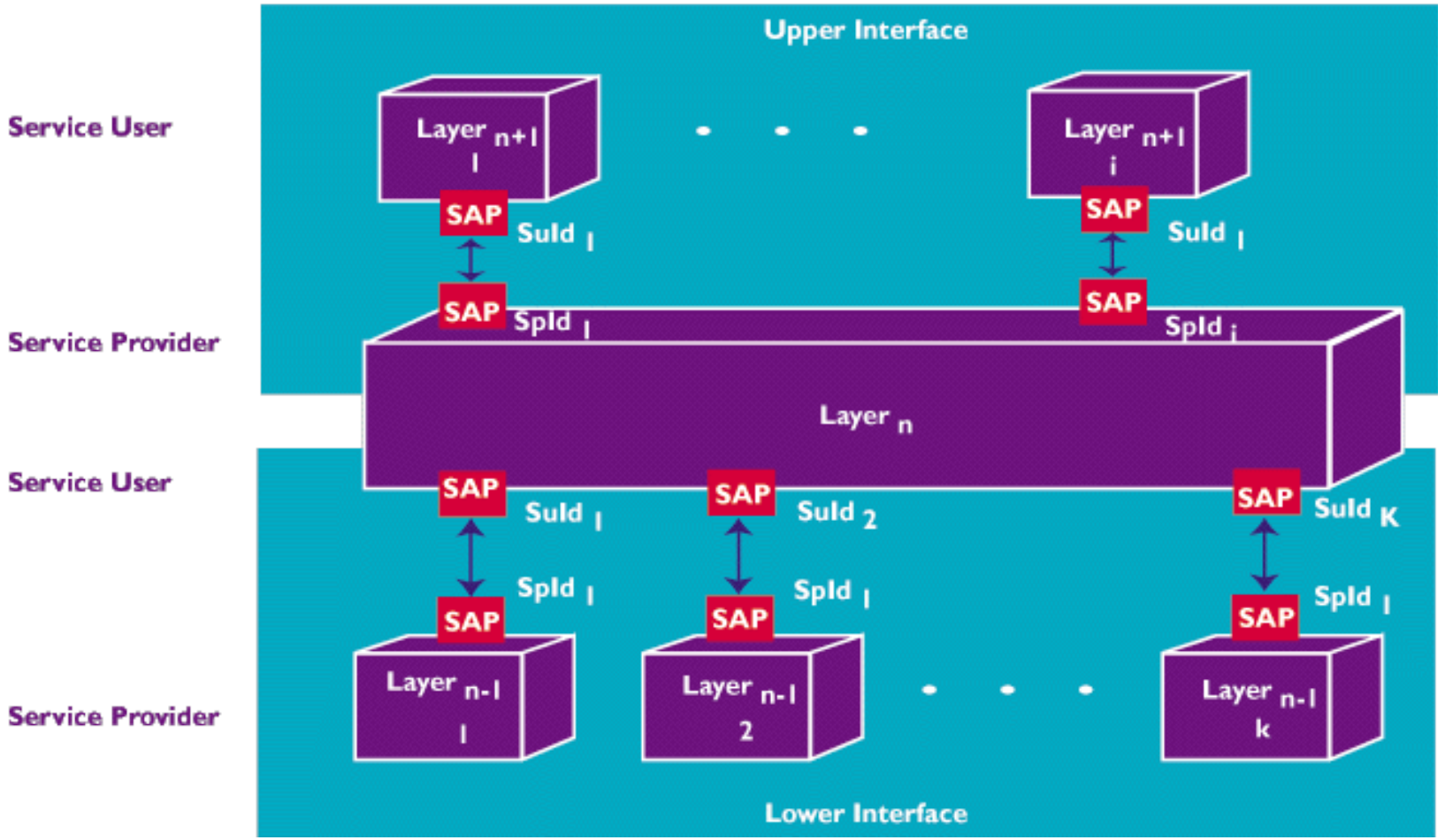


GCC Introduction

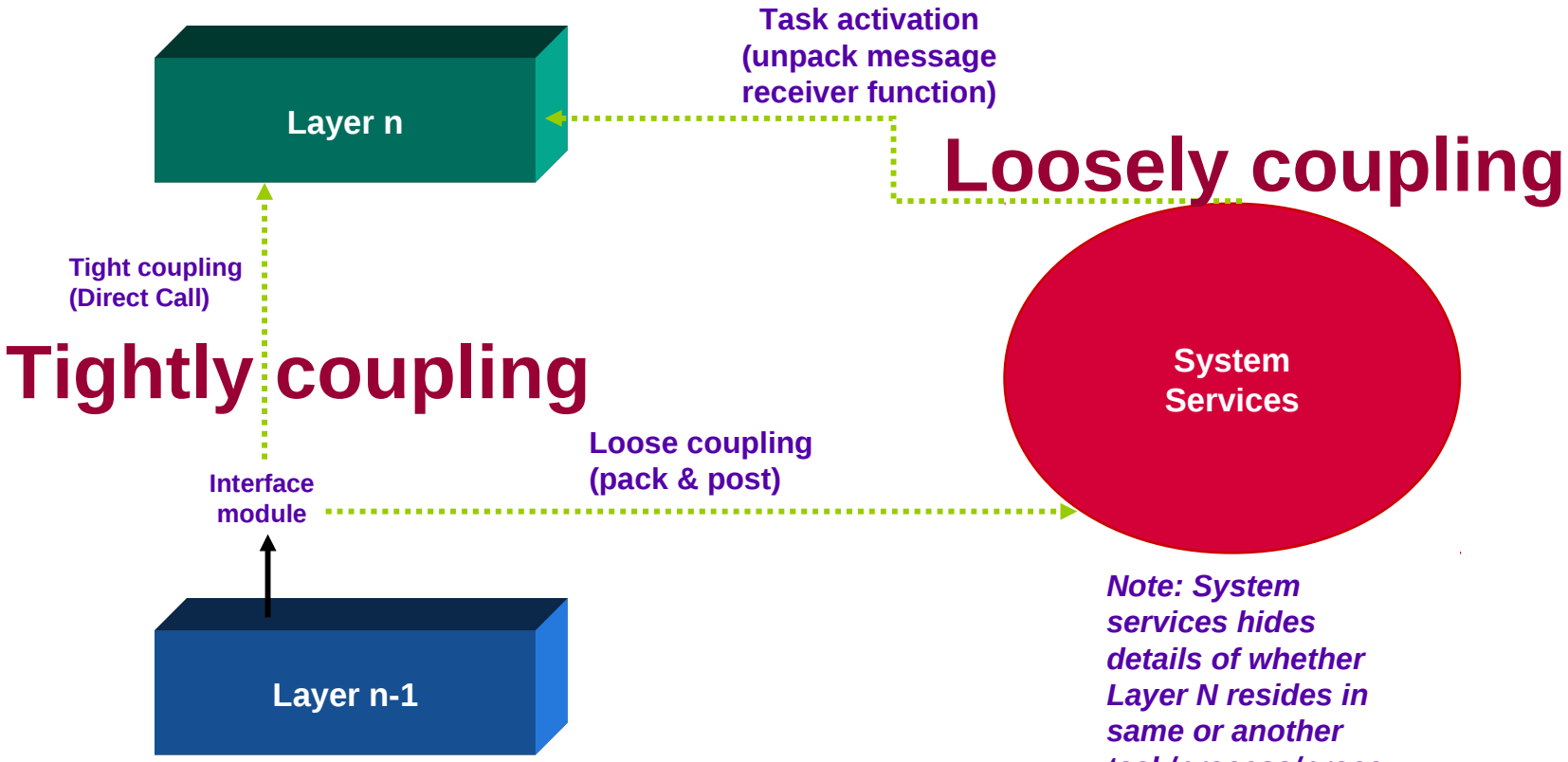
- Call Type
  - Intra-CCS call
  - Inter-CCS call/Inter-CM call
- Internal-BICC
  - As the queue between CCS

GCC Introduction

- Layer Communication



- Plexus Layer Communication



- Plexus Layer Communication
  - Tightly coupling
    - Function Call
  - Loosely coupling
    - IPC
    - Interface can be socket, message queue or shared memory
    - Easily for distributed system

- Plexus Layer Communication
  - Upper/lower layer call

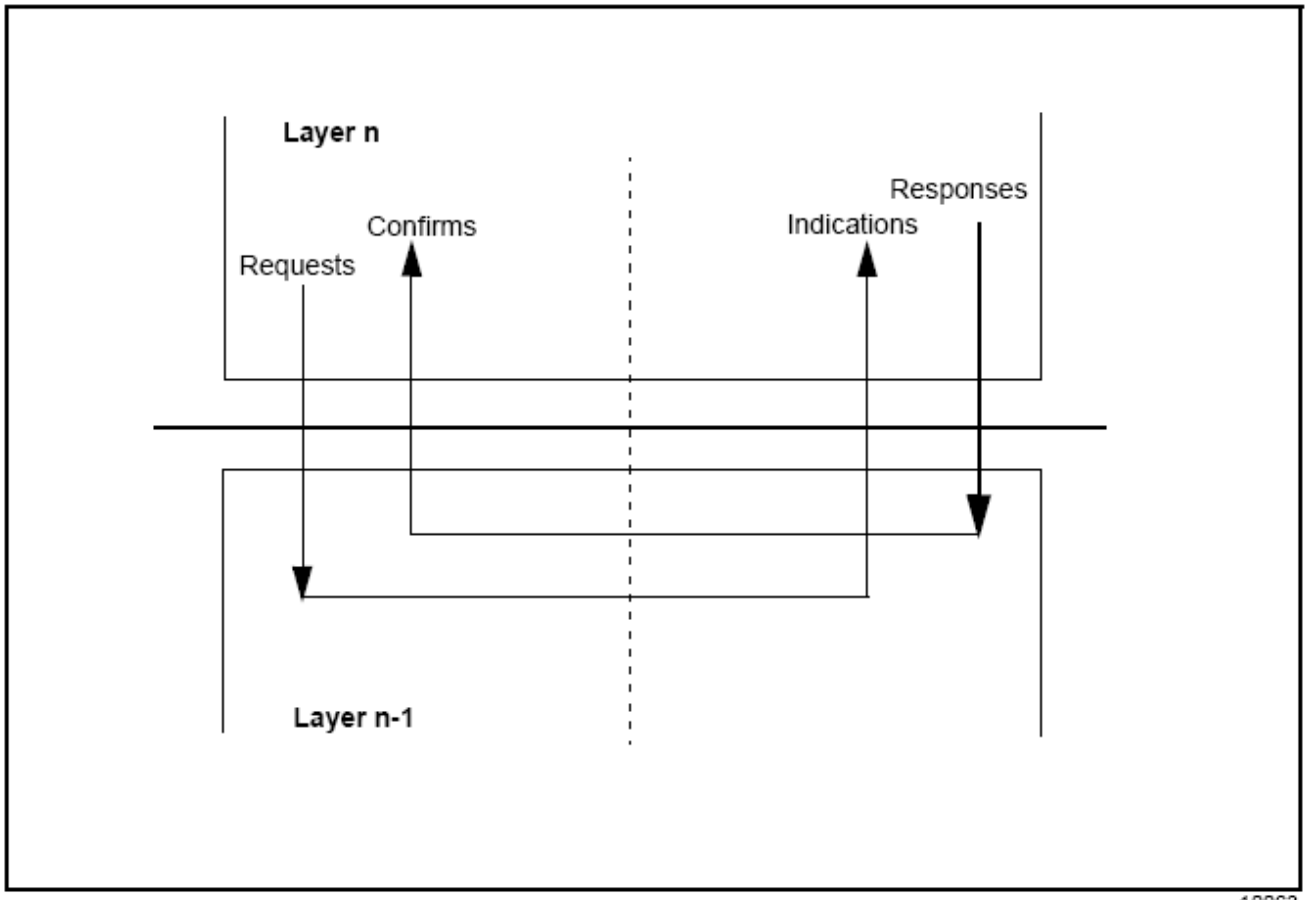
```
functionA(xx, xx)
{
    ……
     ret = (*ccLiCctConReqMt[pst->selector])(pst, spId, suConnId, rsc,
     protType, ccConEvnt, uBuf);
     reEntryFlag = FALSE;
    ……
 }
```

GCC Introduction

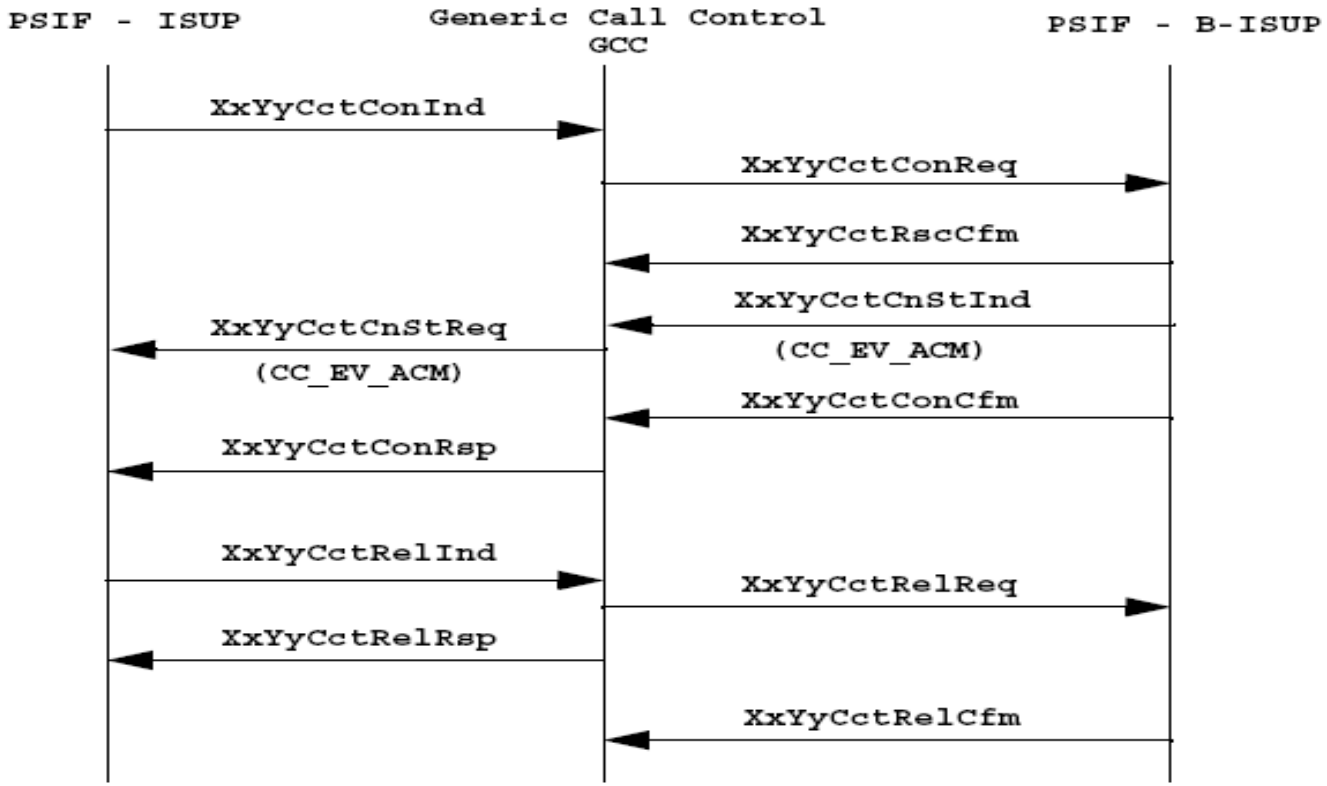
- Plexus Layer Communication
  - Implementation

```
PUBLIC  CONSTANT CctConReq ccLiCctConReqMt [MAXCCLICCT] =
{
  #ifdef LCCCLICCT
   cmPkLiCctConReq,     /* 0 - loosely coupled - fc */
#else
   PtUiCctConReq,       /* 0 - tightly coulpled, portable */
#endif
   PtUiCctConReq,       /* 1 - tightly coupled, portable */
   IwUiCctConReq,       /* 2 - tightly coupled, isup */
   QwUiCctConReq,       /* 3 - tightly coupled, isdn */
   PtUiCctConReq,       /* 4 - tightly coupled, portable */
  ……
}
```

- Plexus Layer Communication Method
  - **Primitives**
    - Requests or responses (from layer n to layer n-1)
    - Indications or confirms (from layer n-1 to layer n)



- Plexus Layer Communication Method
  - **Primitives**
    - Requests or responses (from layer n to layer n-1)
    - Indications or confirms (from layer n-1 to layer n)



- Plexus Layer Communication Method
  - **Interface Naming Conventions**

`<Xx><Yy>Int<Action><Type>`, where,

Name	Description
`Xx`	Specifies the two letter product prefix of the layer. For example, `Si` for ISDN User Part (ISUP).
`Yy`	Specifies whether the primitive is called at a lower interface (`Li`) of the service user, the upper interface (`Ui`) of the service provider, or the layer manager interface (`Mi`)
`Int`	Specifies the name of the interface. For example, `Sit`.
`Action`	Specifies the primitive action. For example, `Bnd` for binding, `Sta` for status.
`Type`	Specifies whether the primitive is a request (`Req`), indication (`Ind`), response (`Rsp`), or confirm (`Cfm`) primitive.

- Plexus Layer Communication Method
  - **Interface Naming Conventions Example**

`XxUiSitConInd` is a connection establishment indication primitive issued by ISUP at its upper interface with the application layer. This can be parsed as:

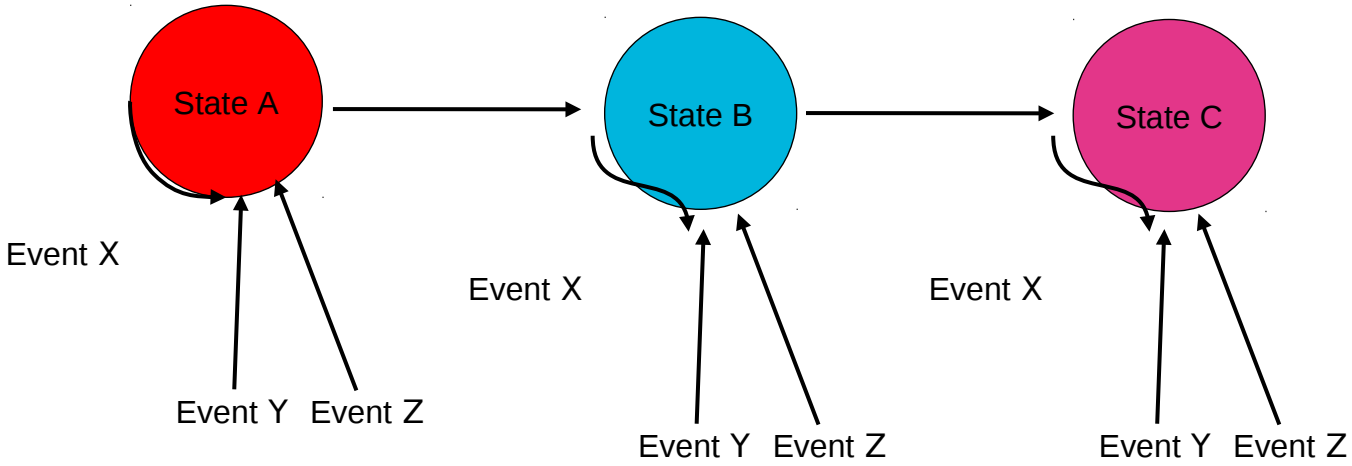
Name	Description
`Xx`	Trillium product prefix for the layer
`Ui`	Upper interface of product
`Sit`	SIT Interface
`Con`	Primitive action is connection establishment
`Ind`	Primitive is of type indication

Agenda

- Plexus Background
- Plexus Architecture
- GCC Introduction
- GCC State Machine
- GCC Protocol Interworking
- GCC Redundancy

GCC State Machine

- GCC State Machine
  - Event driven model



GCC State Machine

- GCC State Machine
  - State Machine **Naming Conventions**
    - ccConEXXSXX
      - Means handle Event XX at State XX.
  - Example

```
*     Fun:   ccConE00S00
*
*     Desc:  Connection state function
*            event - Connection Indication
*            state - IDLE
*     Ret:   ROK     - successful,
*            RFAILED - unsuccessful
**     Notes: None.
*     File:  cc_bdy2.c
*/
```

GCC State Machine

- GCC State Machine
  - State Machine **Table**

```
StateFn stateTable[CCMAXEVENTS][CCMAXSTATES] =
{
    /* Connect indication - 00 */
   {
    ccConE00S00,            /* 00-CCS_IDLE                    */
    ccUnexpEvent,           /* 01-CCS_AWTROUTERSC             */
    ccUnexpEvent,           /* 02-CCS_AWTROUTEDGT             */
   },
   /* Connct confirm  - 01*/
   {
     ccUnexpEvent,          /* 00-CCS_IDLE                    */
   }
}
```

- GCC State Machine
  - Upper/lower layer call

XxYyCctStaInd

→ ccCallStateMachine (con, CCE_CONIND, (PTR)ccConEvnt, suId, spConnId,

(PTR) NULLP, (PTR) NULLP);

→stateTable[evntType][oldstate]

→Mapping Function (If have)

- GCC State Machine
  - Upper/lower layer call

XxYyCctStaInd

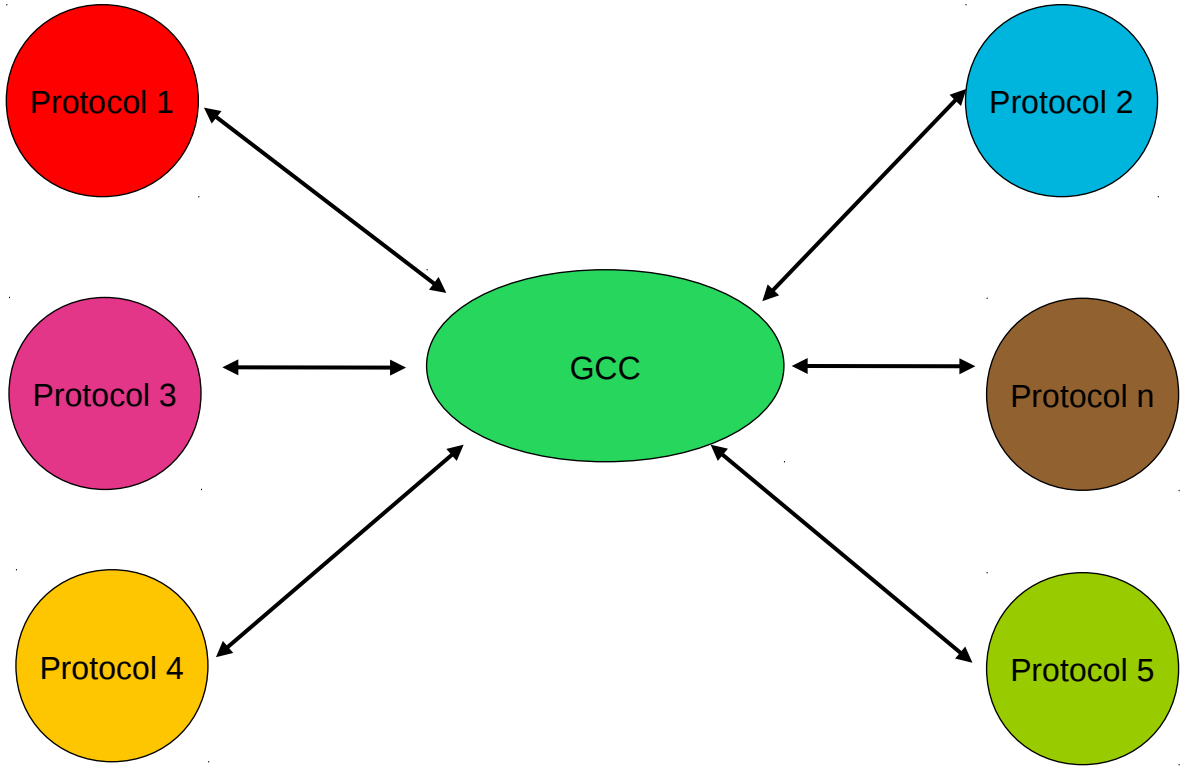
→ ccCallStateMachine (con, CCE_CONIND, (PTR)ccConEvnt, suId, spConnId,

(PTR) NULLP, (PTR) NULLP);

→stateTable[evntType][oldstate]

→Mapping Function ccMapEvent(con, CCE_CONIND, 0); (If have)

- GCC Mapping Function
  - Message Level Mapping for protocol X to protocol Y



GCC Mapping Function

- GCC Mapping Function
  - Get the mapping index for each message
    - Based on incoming and outgoing protocol
    - Based on message direction
    - Based on message type
  - Example

ccGetMappingIndex_real

->ccGetMappingIdx

->SI TO SI

->ccGetNBToNBMappingIdx

->IAM TO IAM

……

or

-> SI TO IN

->ccGetSIToINMappingIdx

……

- GCC Mapping Function
  - After get Map index, call real mapping function
    - Based on incoming and outgoing protocol
    - Based on Mapping index
  - Example

```
ccGetMappingIndex_real
    ->ret = directMappingMatrix[intwIdxIc][intwIdxOg][mapIdx](con);
        ->SIPT_SI_BICC_TO_SIPT_SI_BICC
            ->ccMapS02M00 /* IAM to IAM */
```

GCC Mapping Function

- GCC Mapping Function
  - For message interworking, it is NxN matrix

Direct mapping matrix:

```
PFCCM *directMappingMatrix[MAX_BASE_INTW_PROT][MAX_BASE_INTW_PROT] =
{
   /* INTW_CAS_GR303_LN */
   {
      CAS_GR303_LN_TO_CAS_GR303_LN,/* INTW_CAS_GR303_LN */
      CAS_GR303_LN_TO_CAS_TG,     /* INTW_CAS_TG       */
      ......
   },
  /* INTW_CAS_TG       */
  {
      CAS_TG_TO_CAS_GR303_LN,     /* INTW_CAS_GR303_LN */
     ......
   }
},
```

GCC Mapping Function

- GCC Mapping Function
  - Message mapping function matrix

Direct mapping matrix:

```
PFCCM *directMappingMatrix[MAX_BASE_INTW_PROT][MAX_BASE_INTW_PROT] =
{
   /* INTW_CAS_GR303_LN */
   {
      CAS_GR303_LN_TO_CAS_GR303_LN,/* INTW_CAS_GR303_LN */
      CAS_GR303_LN_TO_CAS_TG,      /* INTW_CAS_TG       */
      ......
   },
  /* INTW_CAS_TG       */
  {
      CAS_TG_TO_CAS_GR303_LN,      /* INTW_CAS_GR303_LN */
     ......
  }
},
```

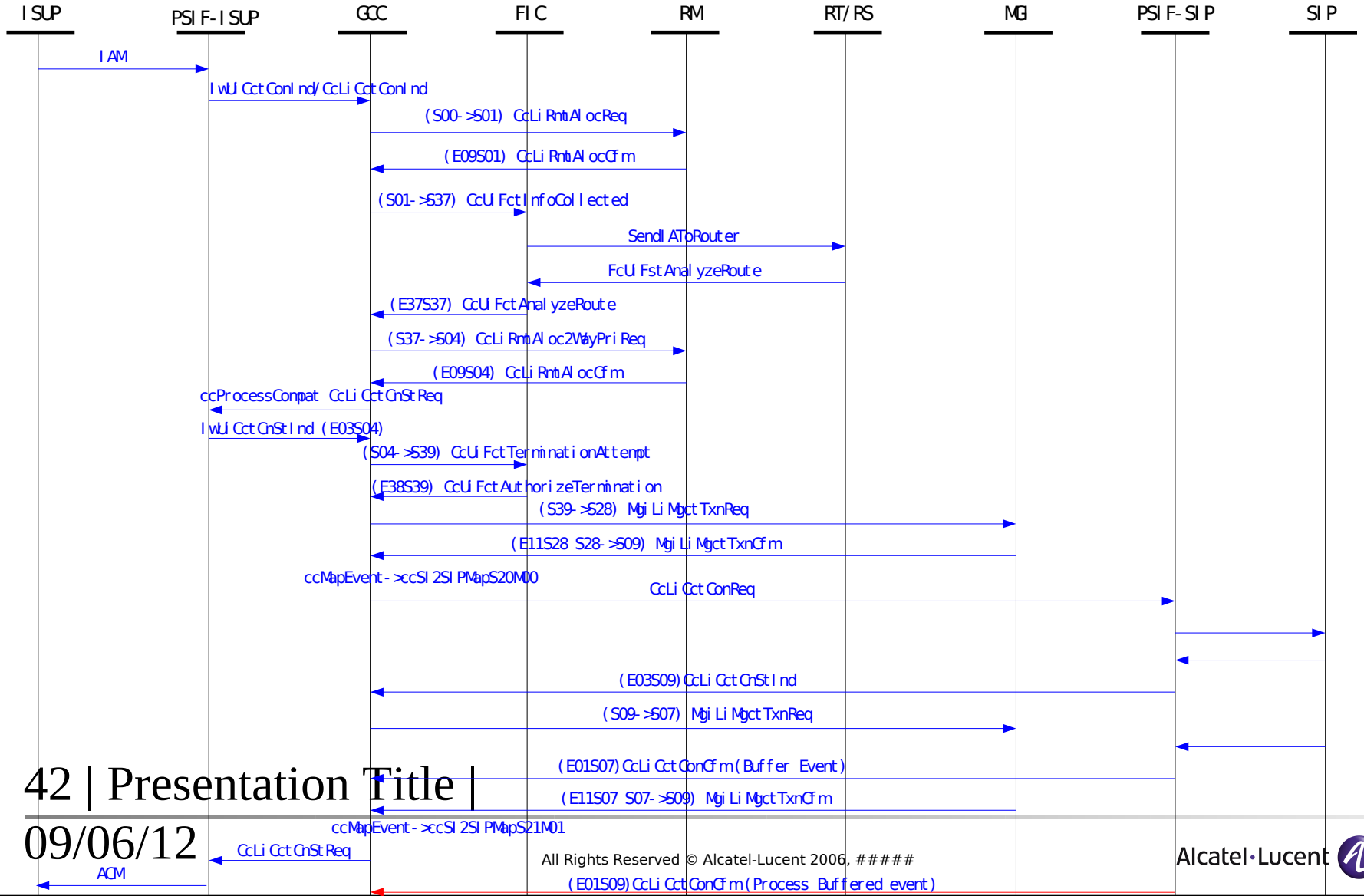
GCC Mapping Function

- GCC Mapping Function
  - For message interworking, it is NxN matrix

```
PUBLIC PFCCM SIPT_SI_BICC_TO_SIPT_SI_BICC[SIPT_SI_BICC_TO_SIPT_SI_BICC_MAX] =
{
   ccMapS02M00,        /* IAM to IAM */
   ccMapS02M01,        /* ACM to ACM */
   ccMapS02M02,        /* CPG to CPG */
   ccMapS02M03,        /* ANM to ANM */
   ccMapS02M04,        /* SUS to SUS */
   ccMapS02M05,        /* RES to RES */
   ccMapS02M06,        /* SAM to SAM */
   ccMapS02M07,        /* FOT to FOT */
  ……
 }
```

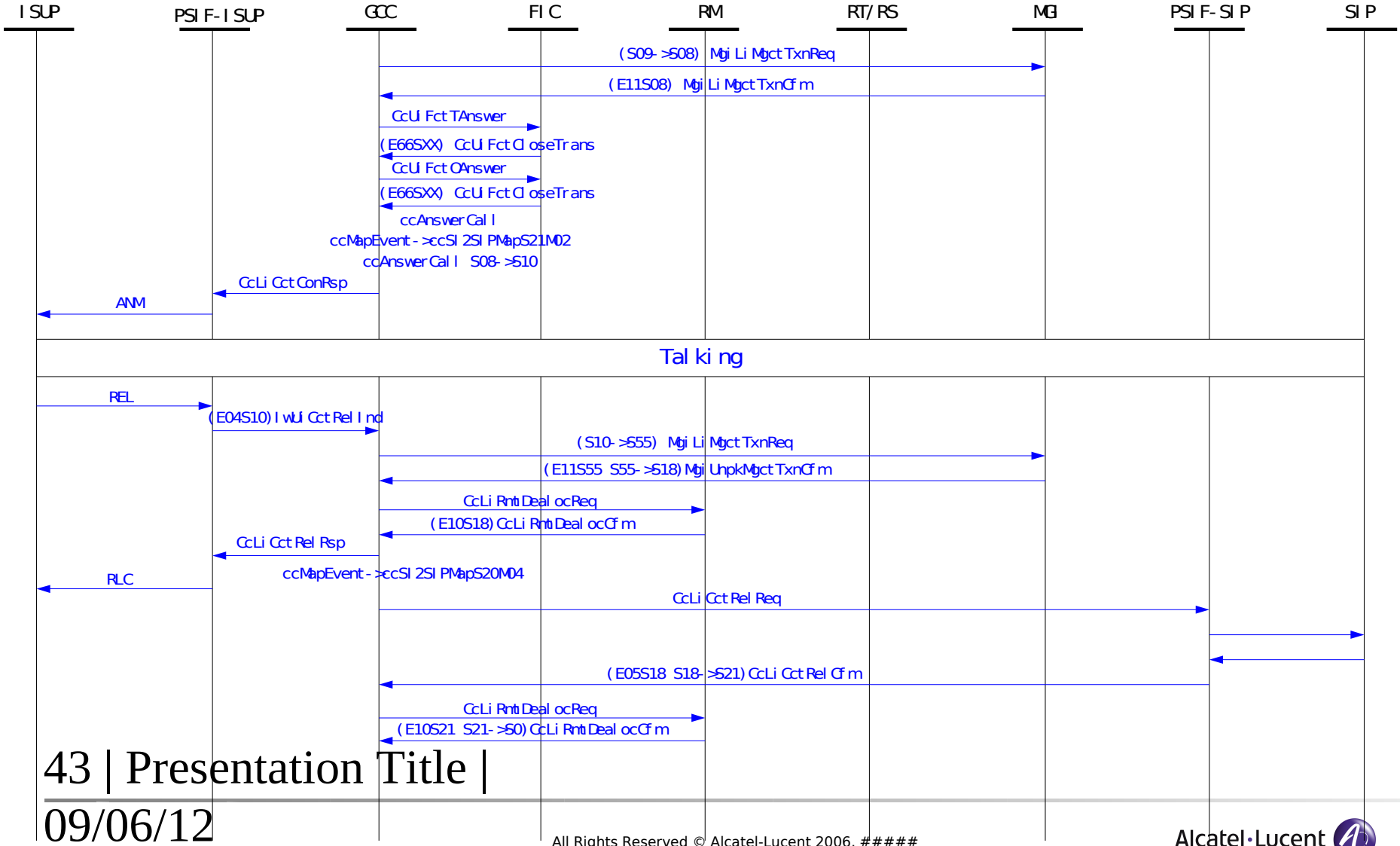
GCC Mapping Function

- GCC Basic Call Flow



GCC Mapping Function

- GCC Basic Call Flow Cont



GCC Mapping Function

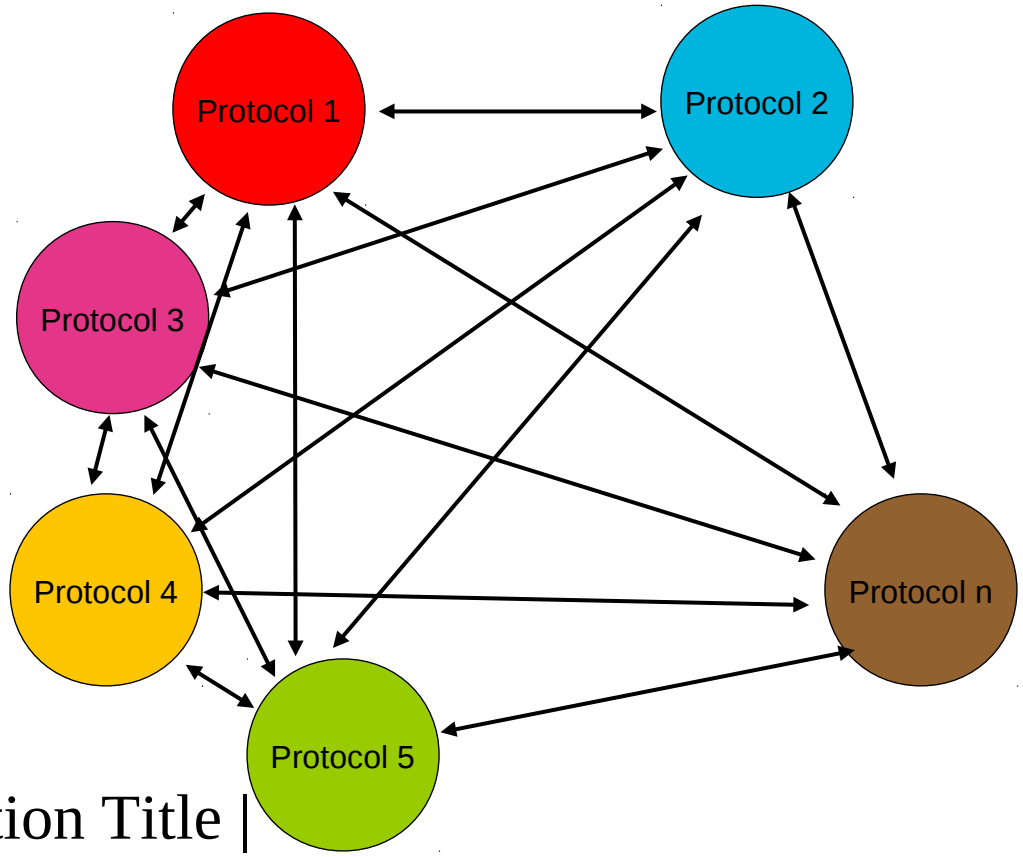
- GCC Basic Call Flow Cont
  - Reference: ..\..\..\work\backup\new work\china isup\plexus-callflow.vsd

Agenda

- Plexus Background
- Plexus Architecture
- GCC Introduction
- GCC State Machine
- GCC Protocol Interworking
- GCC Redundancy

GCC Protocol Interworking

- Direct Mapping
  - Traditional Protocol Interworking method
  - Protocol X direct mapping to Protocol B

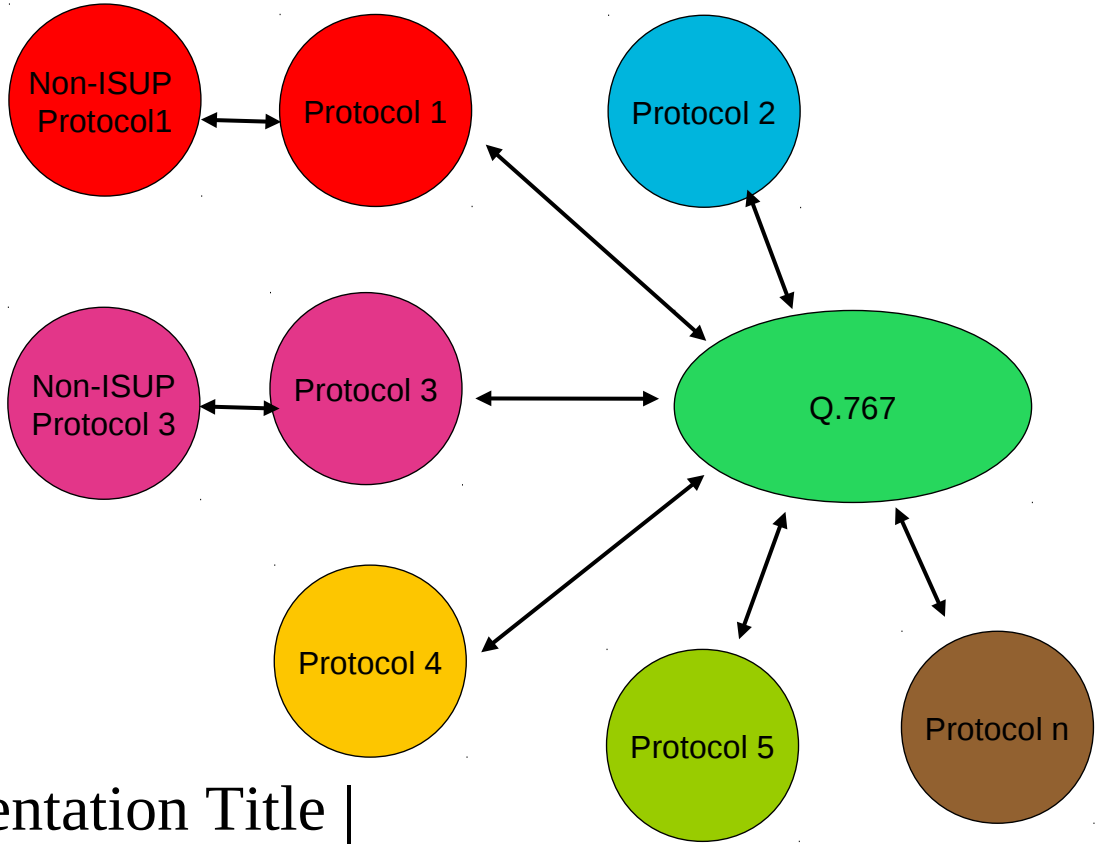


GCC Protocol Interworking

- Direct Mapping Analysis
  - Mapping Path
    - Interworking path number will increase non-linearly
    - The interworking path number will be 1/2n(n-1) for n protocol
  - Standards
    - No specific stands for all interworking path
  - Effort
    - Adding one country, N-1 interworking path will be added
    - One country variant parameter changed, N-1 interworking need to be considered
    - Testing, development and maintenance effort will be huge
  - Reuse
    - None existing interworking path can be reused
  - System Stability
    - System stability will be highly decreased

GCC Protocol Interworking

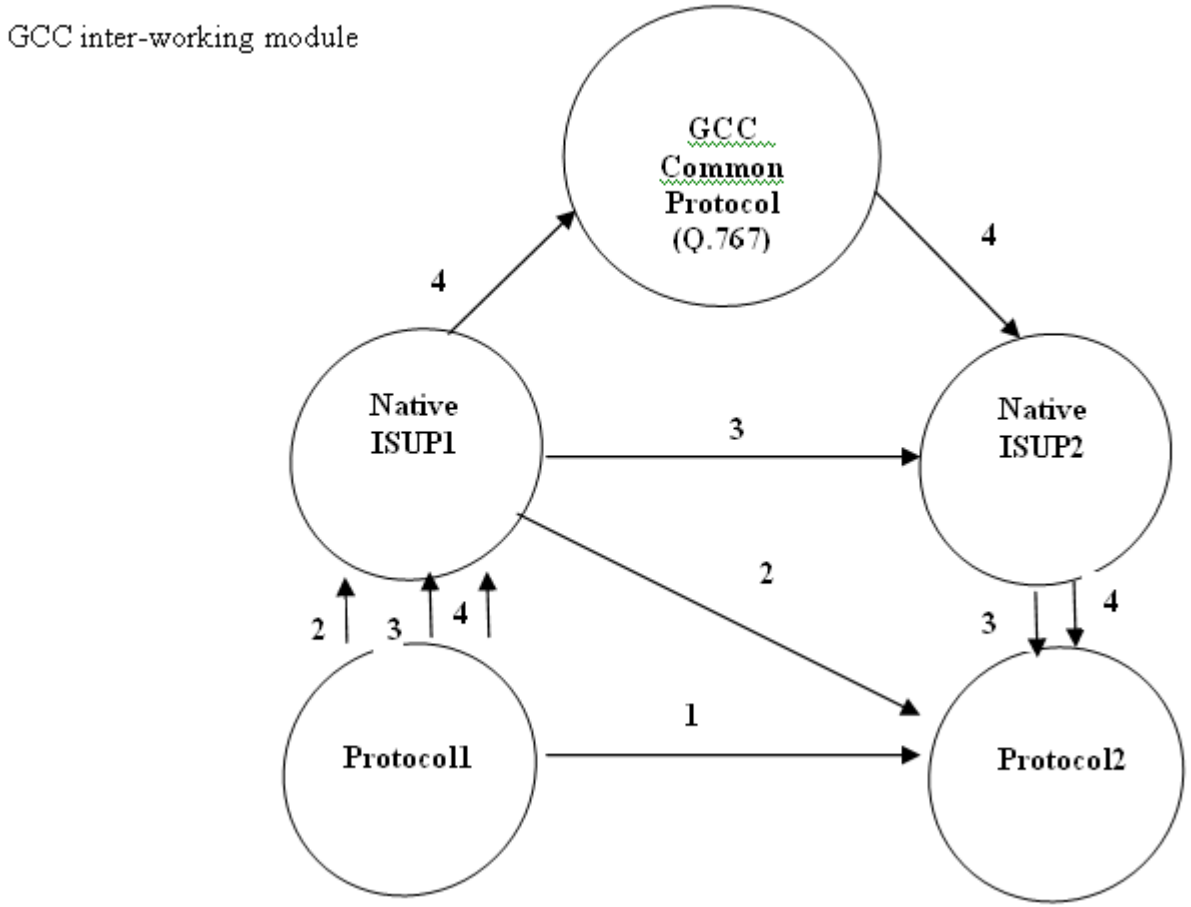
- Indirect Mapping
  - Plexus new interworking method, introduced from 6.2.1
  - Protocol X mapping to Protocol B via common protocol



- Indirect Mapping Analysis
  - Mapping Path
    - Interworking path number will increase linearly
    - The interworking path number will be n for n protocol
  - Standards
    - Just need to find one standard for each country and common protocol
  - Effort
    - Adding one country, only 1 interworking path need to be added
    - One country variant parameter changed, only 1 interworking need to be considered
    - Testing, development and maintenance will be easy
  - Reuse
    - Can reuse all the existing interworking path
  - System Stability
    - System stability almost no change

GCC Protocol Interworking

- GCC Indirect Mapping



GCC Protocol Interworking

- GCC Indirect Mapping

  ccMapEvent_real

  ->ccGetMapTblEntry

  checking mapping table to find is protocol A and B direct mapping or not

  Yes, follow old logic

  NO ->ccComputeIntwPath

  - Example
    - ISDN<-->China ISUP<-->enhanced Q.767<-->ANSI ISUP<-->SIP

GCC Protocol Interworking

- GCC Indirect Mapping Table
  - Native ISUP Table

```
PUBLIC Void ccInitNatIsupTbl
(
Void
)
{
    /* CC_IN */
    CC_INSERT_NATISUP(CC_INETSI, CC_SIETSI);
    CC_INSERT_NATISUP(CC_INNI2, CC_SIANS92);
    CC_INSERT_NATISUP(CC_INITU, CC_SIITU92);
    CC_INSERT_NATISUP(CC_CS_LN, CC_SIANS92);
    CC_INSERT_NATISUP(CC_CS_TG, CC_SIANS92);
    ……
}
```

GCC Protocol Interworking

- GCC Indirect Mapping Table
  - SI Mapping Table

```
PUBLIC Void ccInitSI_TO_MapTbl
(
Void
)
{
    CC_INSERT_MAPTBL(CC_SIITU92, CC_SIITU92);
    CC_INSERT_MAPTBL(CC_SIITU92, CC_SI76792);
    CC_INSERT_MAPTBL(CC_SI76792, CC_SIITU92);
    CC_INSERT_MAPTBL(CC_SI76792, CC_SI76792);
    CC_INSERT_MAPTBL(CC_SI76792, CC_SIANS92);
    CC_INSERT_MAPTBL(CC_SI76792, CC_SIETSI);
    CC_INSERT_MAPTBL(CC_SI76792, CC_SIFTZ);
    ……
}
```

Agenda

- Plexus Background
- Plexus Architecture
- GCC Introduction
- GCC State Machine
- GCC Protocol Interworking
- GCC Redundancy

- GCC Redundancy
  - Compact switch
    - Each SP has one CCS
    - CCS fault cause SP failover
  - Distributed switch
    - Each CM has 8 CCS
    - Any CCS fault cause CM failover
  - In General, GCC is card level redundancy, not process
  - Call processing data will replicate to standby after talking.
  - Configuration data stored in DB handled by DB redundancy

GCC Redundancy

- GCC Hot Upgrade
  - Cold Upgrade
    - All standing call will fail
    - All new attempt call will fail
  - Hot Upgrade
    - All standing call will no impact
    - Part of new attempt call will fail (Very limit)

- GCC Hot Upgrade
  - Why can hot upgrade
    - First upgrade standby card with new version load
    - The old version master load, replicate data to new version
    - Standby load check incoming data version
      - Same with own – no more action
      - Different version – call the related unpack function
      - For new data, set to initialize value
    - After new version is ready
      - Switch over
      - Upgrade the other old load
      - Two side version is same

GCC Redundancy

- GCC Hot Upgrade
- Example

```
PRIVATE S16 zcUnpkRtCreateConCb(pst, tabType, updType, mBuf)
{
   ……
  if (version == peerVersion)
  {
       zcUnpkStructFn(pst, tabType, updType, size, (PTR)&createConCb, mBuf)) != ROK)
  }
  else
  {
     size = sizeof (ZcRtCreateConCb_v_1_1);
     if ((ret = zcUnpkStructFn(pst, tabType, updType,
                        size, (PTR)&createConCb_v_1_1, mBuf)) != ROK)
  }
```

Question & Answer

Thanks