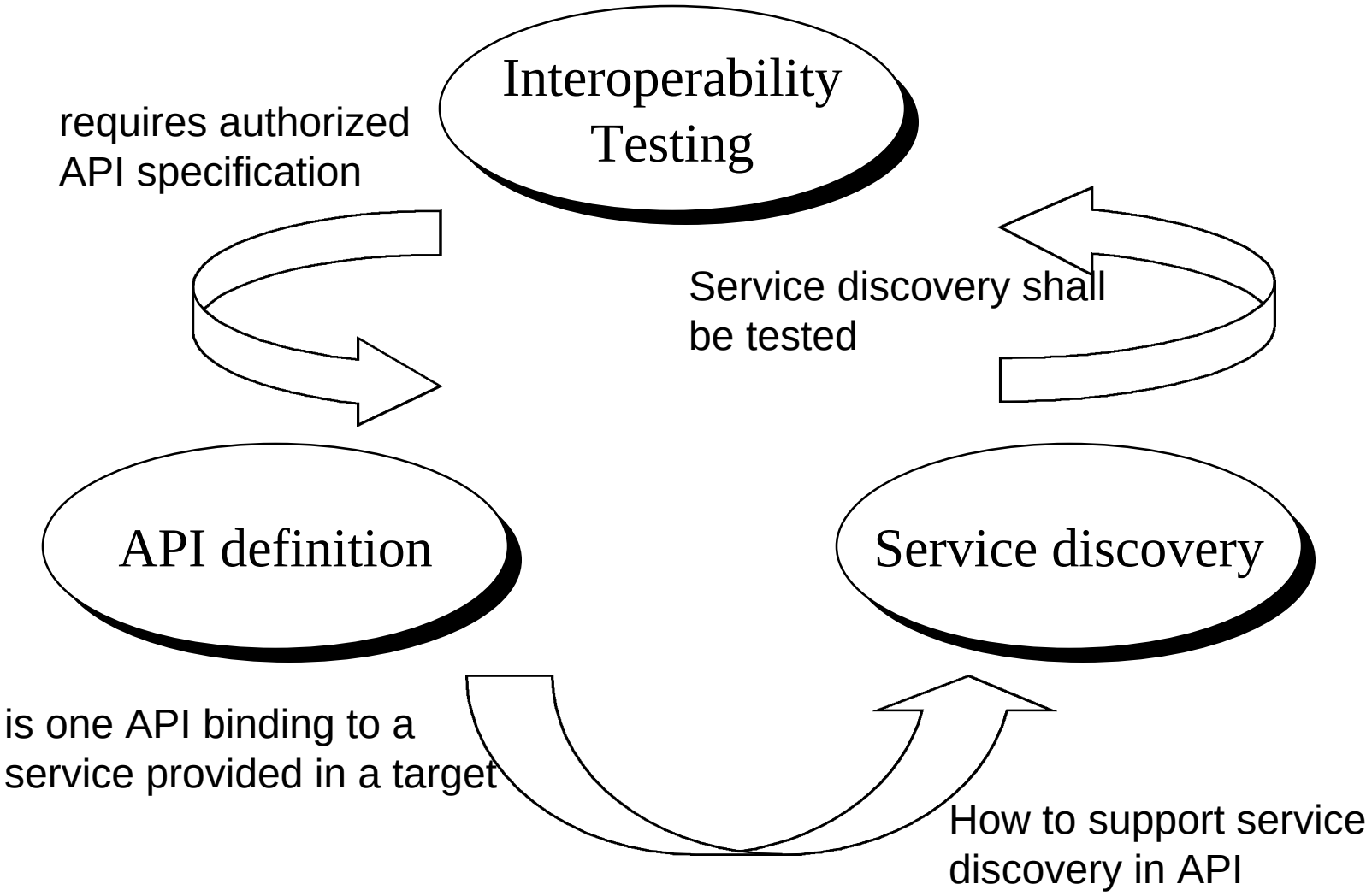
Introduction for

SPI mapping
Service Discovery
Interoperability Testing

20, Sep. 1999 1394 PWG

Fumio Nagasaka

Open Issues are related to each other



Service Provider Interface mapping for PPDT

Fumio Nagasaka

Seiko Epson

20, Sep. 1999

Why API ?

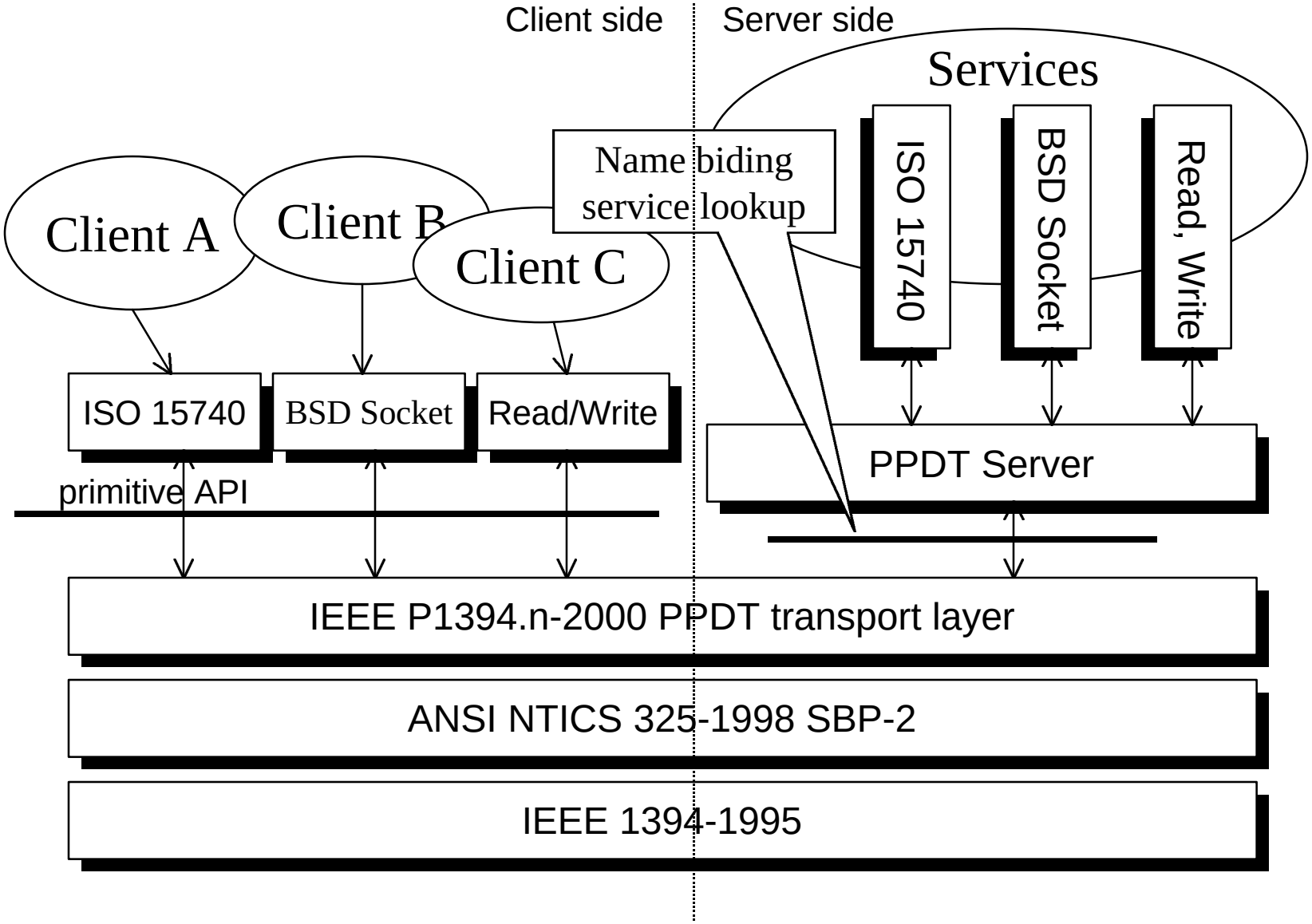
- Still API definition is an open issue
- Clients use API
- Interoperability testing requires authorized API

API policy, Type A

- API only includes few atomic functions
- Layered wrapper class satisfies more complex requirements from a client

- Pros
  - simple
- Cons
  - layered overhead and redundancy

Example of Type A

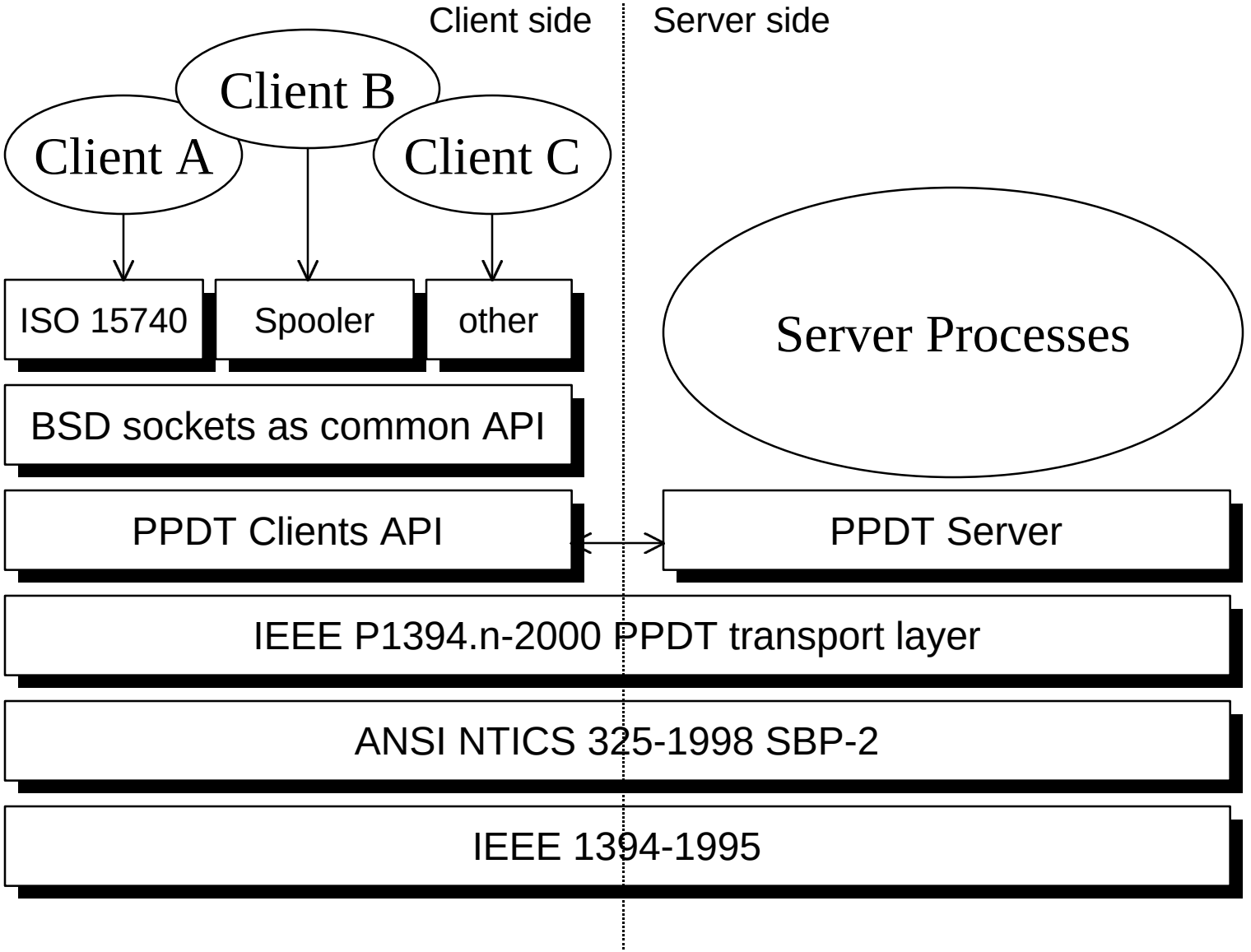


API policy, Type B

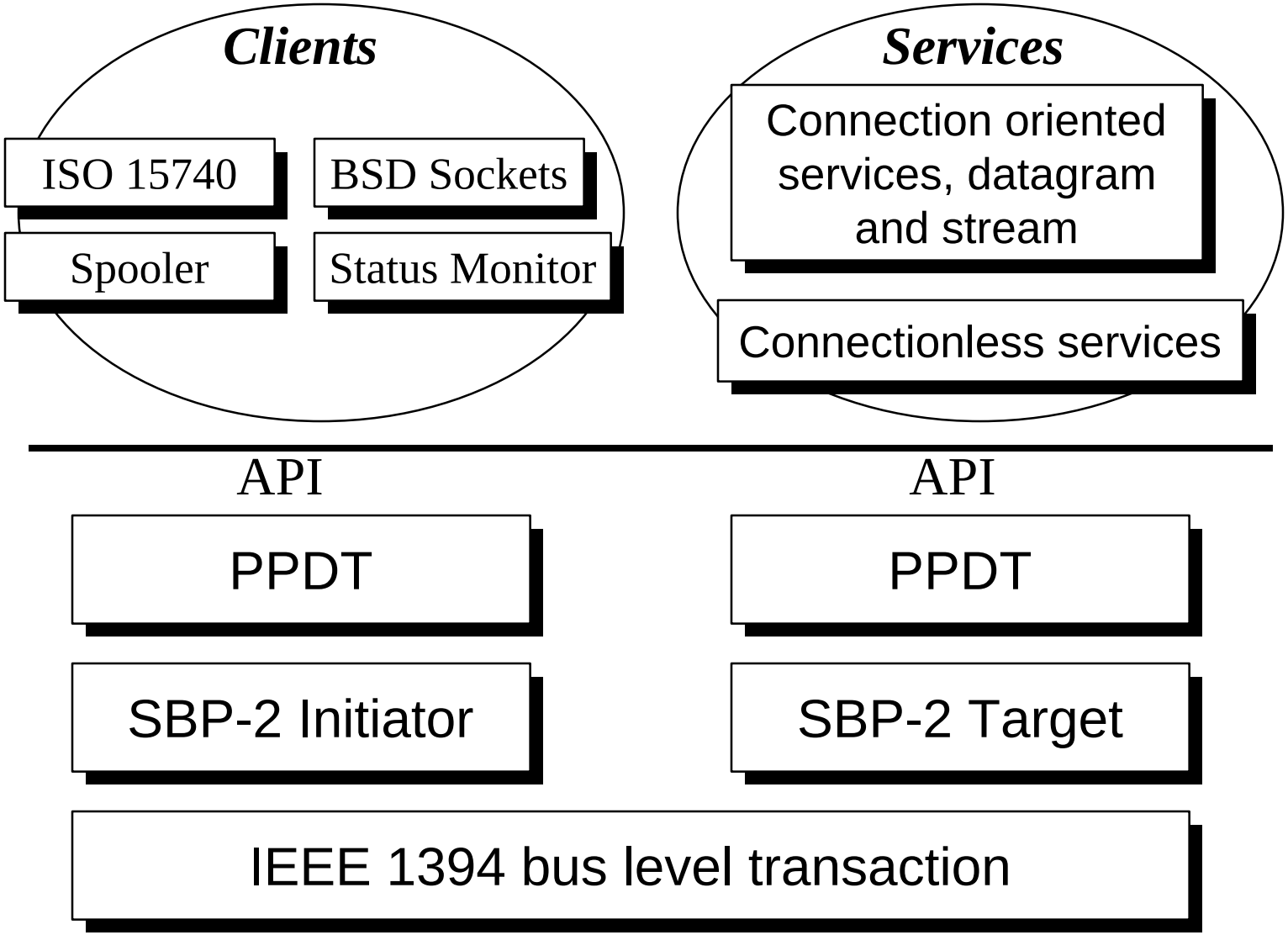
- well examined traditional set of functions
- common access from several types of clients (for example BSD sockets?)

- Pros
  - reusability and compatibility with existing protocol
- Cons
  - every common set has exceptions

Example of Type B

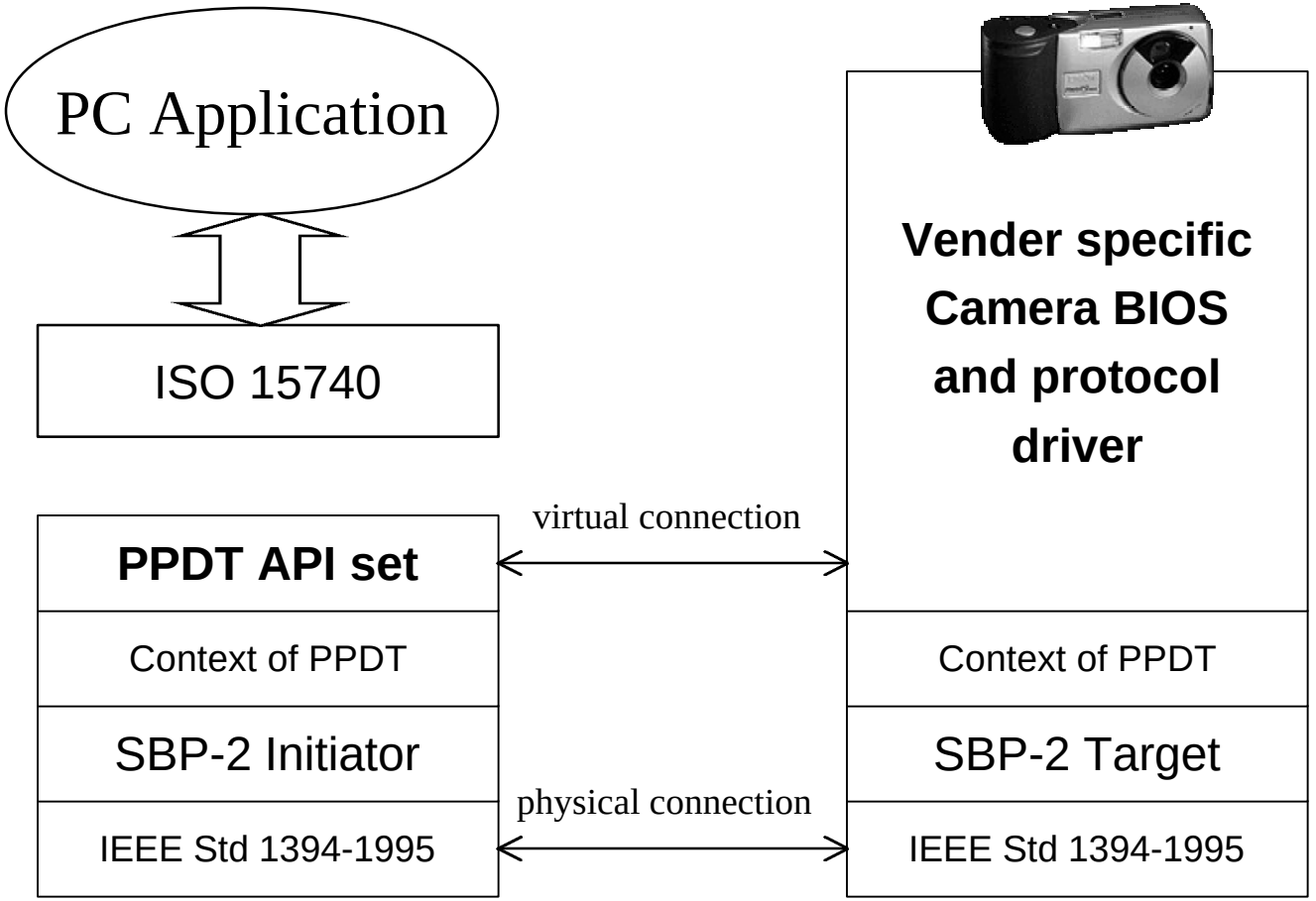


what shall ppdt-API provide?



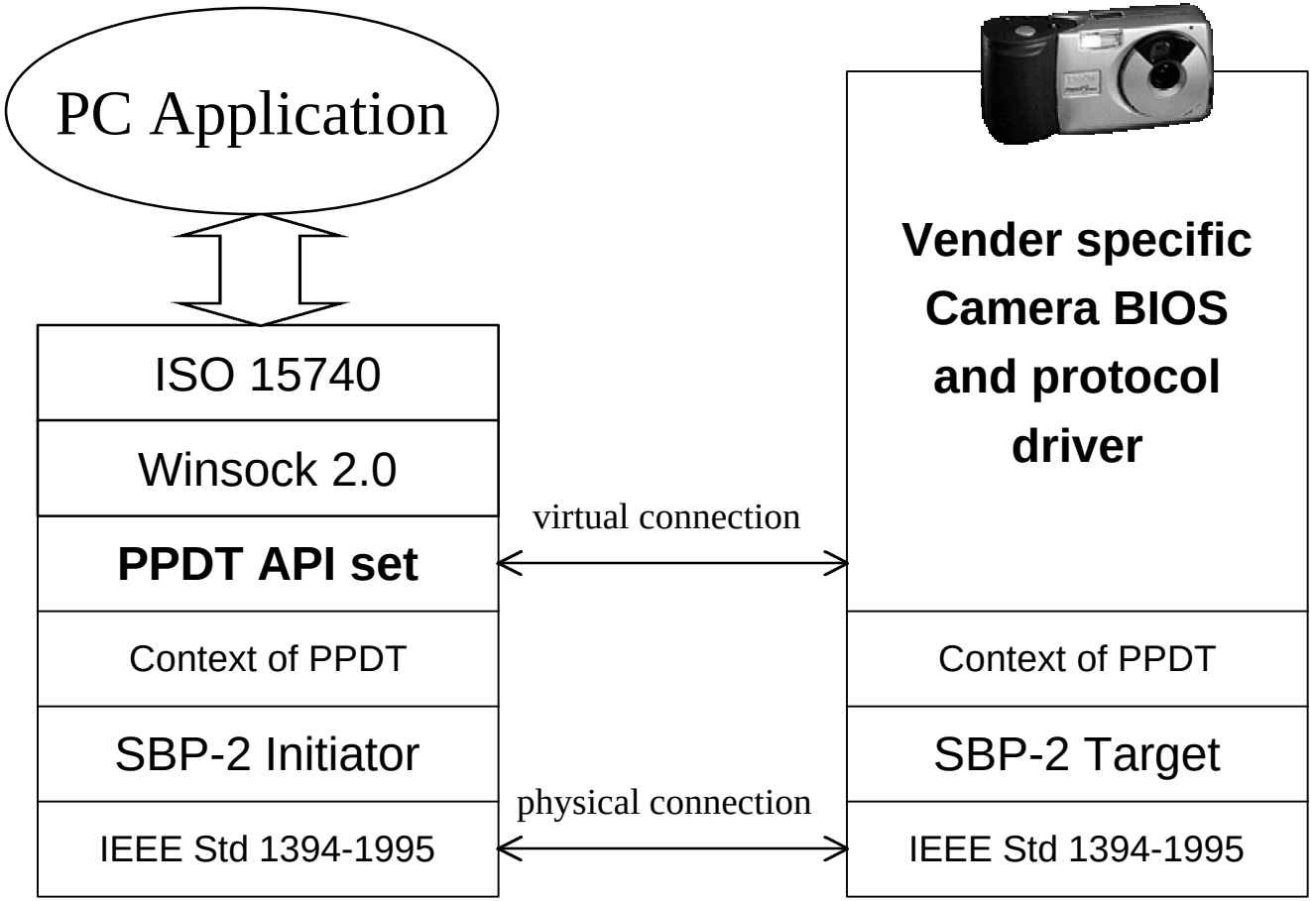
Example 1

– Camera may use ISO 15740(PTP) directly over PPDT



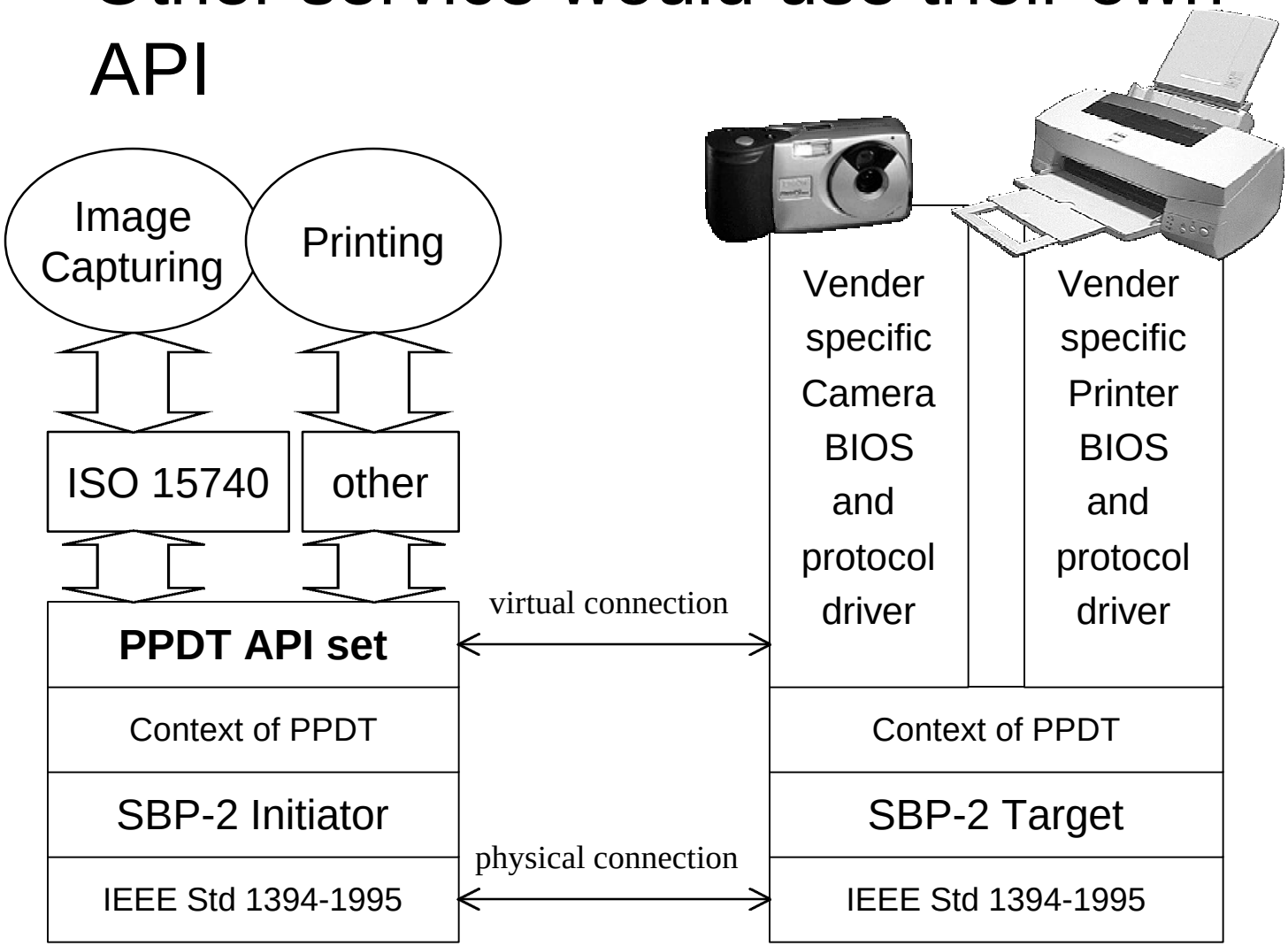
Example 2

– Camera may use ISO 15740(PTP) over common API like Winsock 2.0



Example 3

– Other service would use their own API



Abstraction

- Application shall be abstracted from a transport

PWG shall assume four clients

- Print Spooler
- Status Monitor
  - normally connected services with read/write/open/close style API
- ISO 15740
  - connection oriented session layer requires {event, operation, data transfer and response}, PC2001 requirements for image input device
- BSD Sockets

Services and Clients

- Symmetry of PPDT allows each peer to start session
- Symmetry of PPDT requests both PPDT at SBP-2 target and PPDT at SBP-2 initiator to provide same API for their clients process

Proposal

- { PPDT server at a target, PPDT server at an initiator, PPDT client at a target and PPDT client at an initiator } require different design, consequently these are different contexts.

- To facilitate these contexts to have common API for client applications, ***make wrapper class for each context***.
- Common API shall ***be Socket Style API***.
- Address family shall be ***AF_1394PWG***.

Proposal (continued…)

- PPDT server implementation shall provide service-name registry
  - PPDT context shall not have static name string programmed in source code, but PPDT context shall have internal name space known as name registry.
- Client processes of PPDT server may register service name utilizing API

Address family registration

- **The 1394 PWG shall contact Windows Socket 2 Identifier Clearinghouse.**

- **Address families are shown from,**
  - http://www.stardust.com/wsresource/winsock2/ws2ident.html

Case Study for BSD Socket

following up Brian’s idea

Socket API

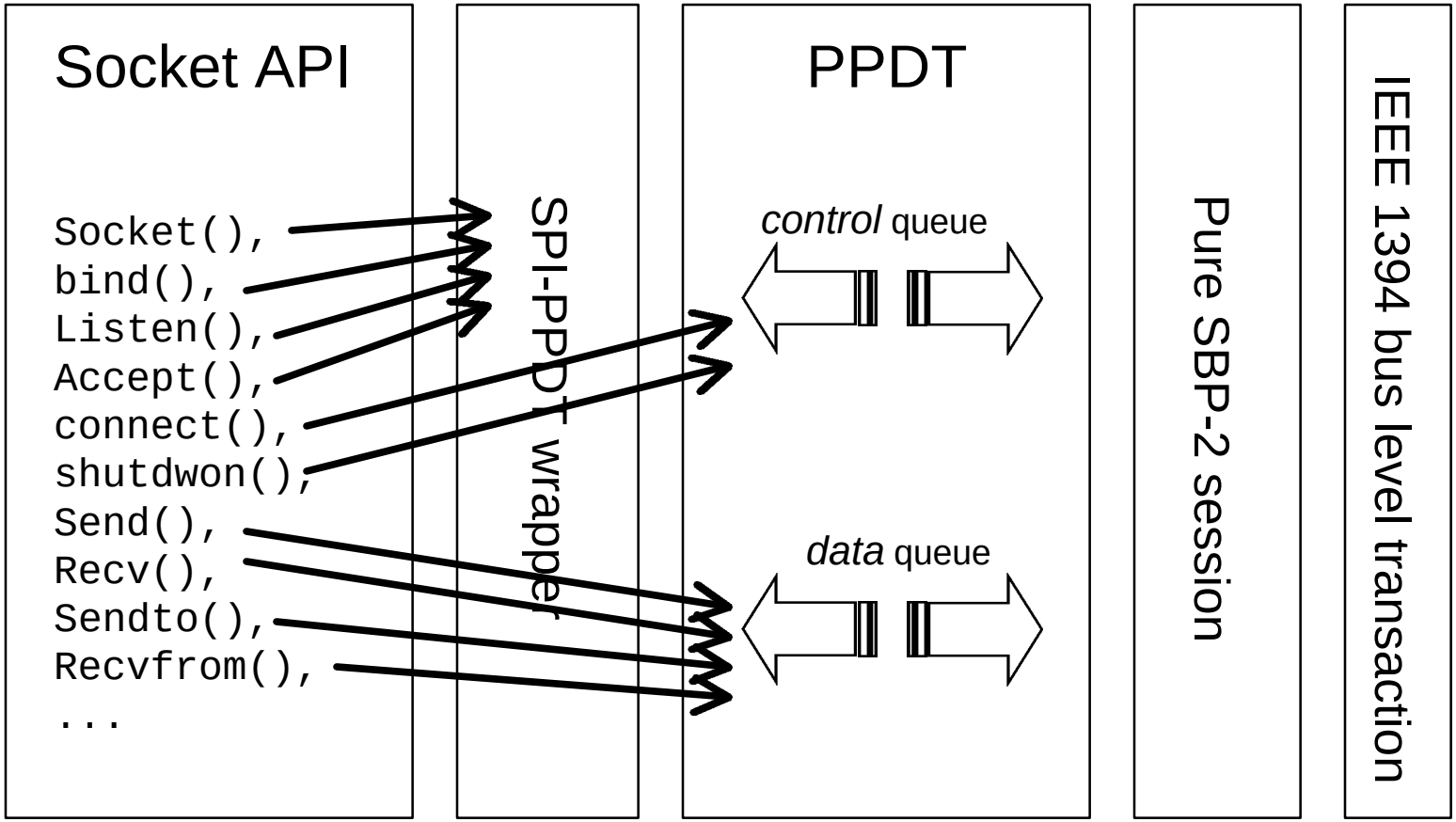
- BSD UNIX Sockets normally requires association of the set of
  - { ***protocol, local-addr, local-process, foreign-addr, foreign-process*** }
- PPDT requires association of the set of
  - { ***mode, local-addr, client-application, foreign-addr, service*** }

PPDT facilities

- PPDT provides two types of queue
  - Control queue
    - *queue* == 0
    - Open/Close uses management queue
  - Data transaction queue
    - *queue* <> *0*
    - Read/Write uses data transaction queue

Service Provider I/F wrapper

- *SPI requires wrapper layer to map each functions to PPDT queues*



Socket

int socket(int $family$, int $type$, int $protocol$);

- each context of PPDT peer internally allocate working memory and build task set.

Listen

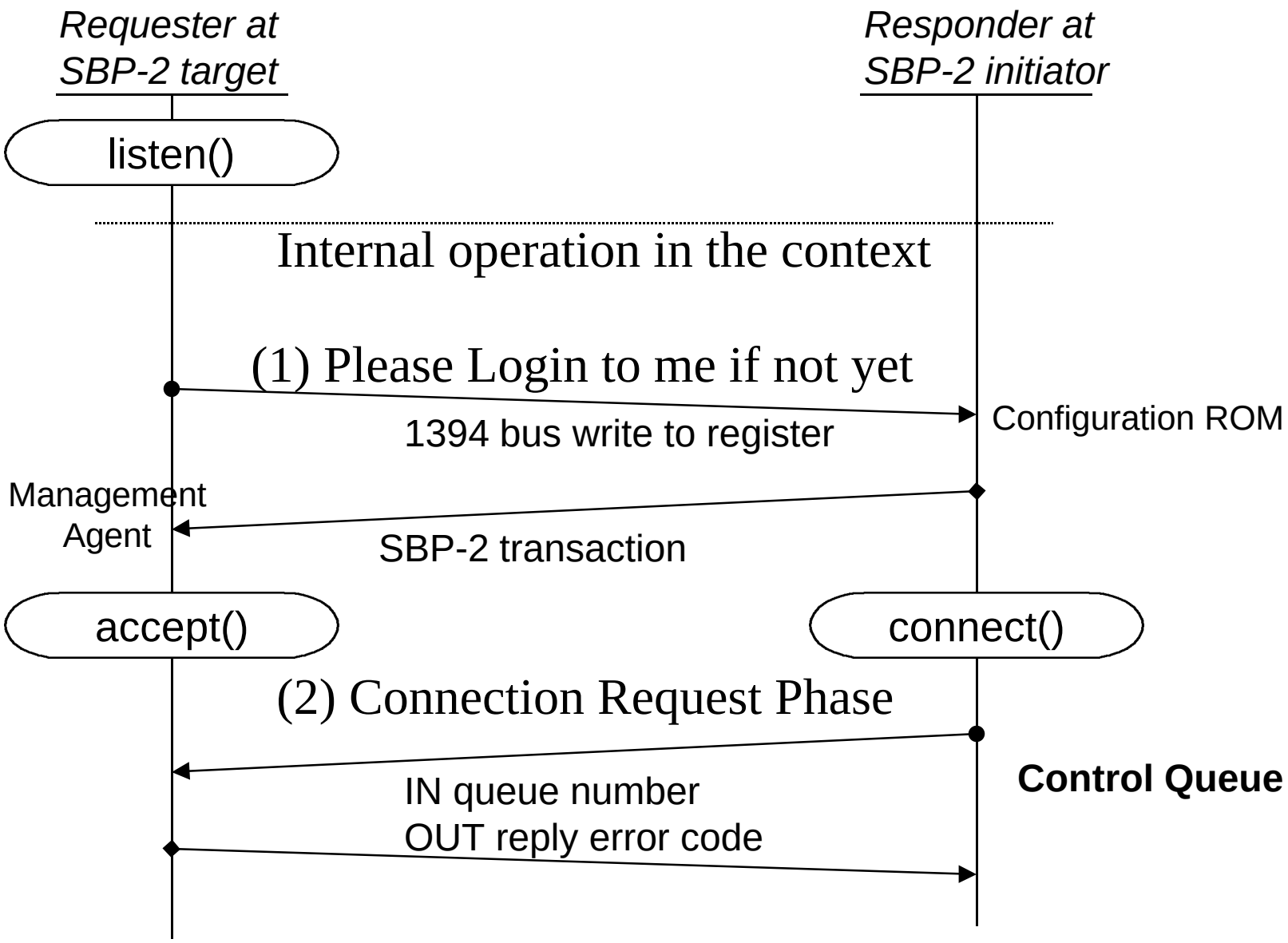
int listen(int *sockfd*, int *backlog*);

sockfd : socket descriptor,

backlog : maximum ***task set*** *size.*

- PPDT server on SBP-2 initiator tries to LOGIN to the target device, if not yet logged in,
- PPDT server on SBP-2 target indicate the initiator to solicit LOGIN, if yet the initiator has not logged in.

Listen (continued …)



Accept

int accept(int *sockfd*, struct sockaddr **peer*, int *addrlen*);

sockfd : socket descriptor,

peer : address of the connected peer,

addrlen : length of the data structure listed in previous line .

Accept (continued …)

- PPDT server on SBP-2 initiator provides control queue to accept “connect()” request,

- PPDT server on SBP-2 target initialize the fetch agent for the control queue.

Bind

- the server of connection oriented transport requires bind() to register the mapping for a service descriptor and a queue number.
- The client and the server of connectionless transport also requires to call bind() by the same reason.

Bind (continued …)

Client application of a server calls this function to map service name and queue number in its name space.

int bind(int *sockfd*, struct sockaddr **myaddr*, int *addrlen*);

sockfd : socket descriptor,

myaddr : protocol specific data structure for address,

addrlen : length of data structure specified in previous line,

Bind (continued …)

protocol specific data structure requires;

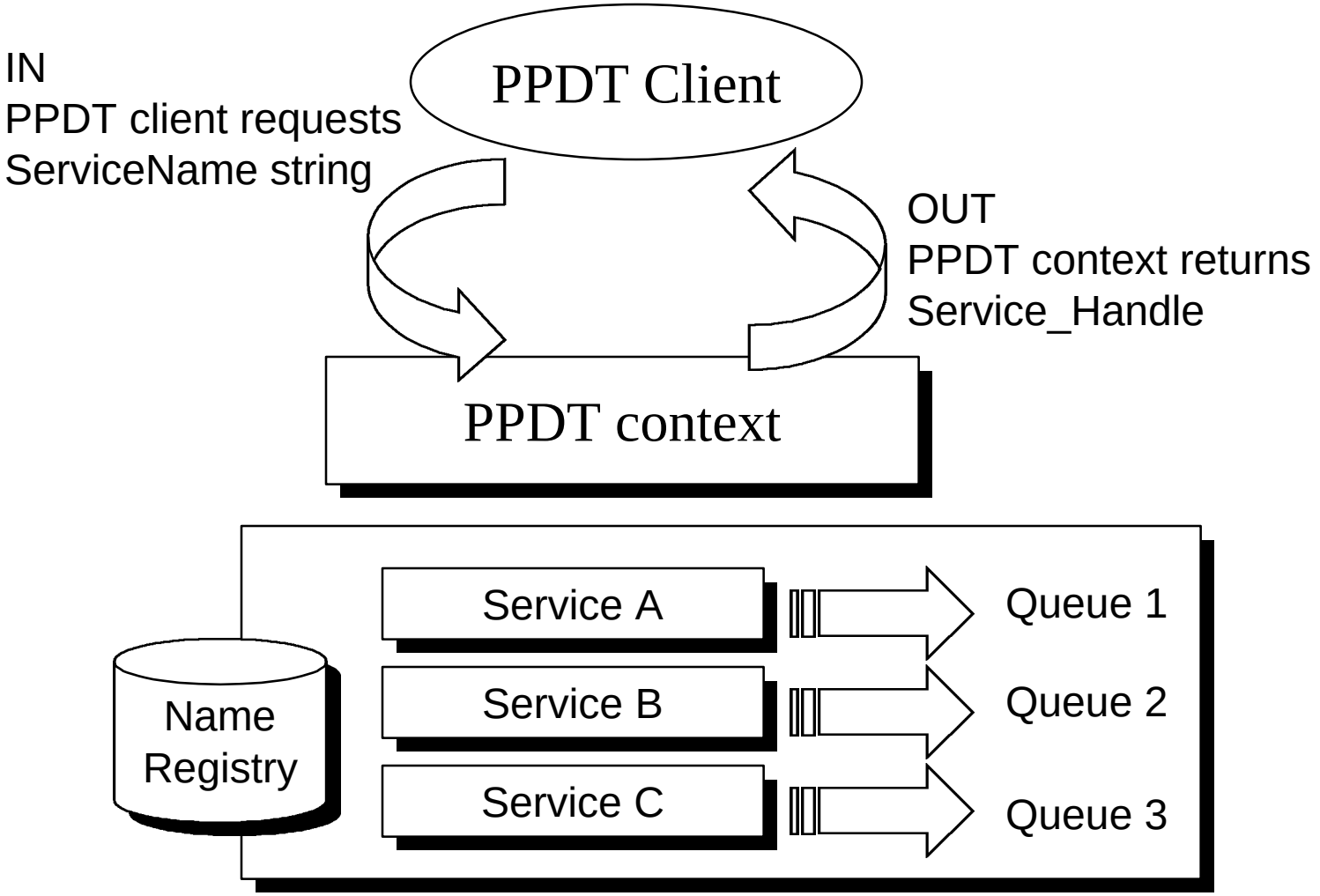
{ service handle, service name, node GUID }

- the pair of [service handle] and [service name] is internally registered and stored in name space of the server.
- the implementation of “Bind()" for a PPDT at a target and a PPDT at an initiator might be same.

Proposal

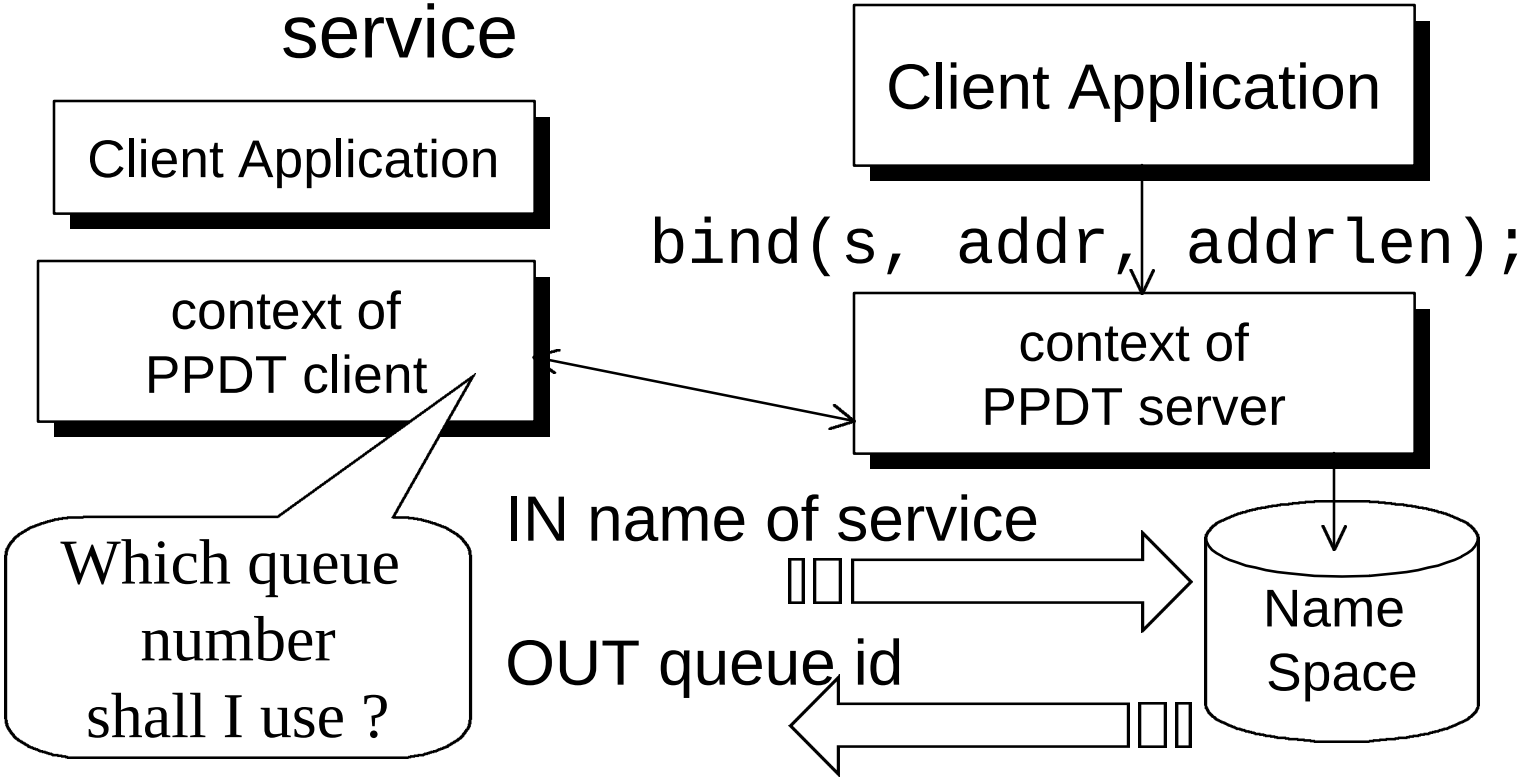
- PPDT client specifies the ASCII string of required service
- PPDT context returns the service handle of associated service
- The service handle shall be associated internally for queue id
- The context shall keep association described above

Proposal *(continued …)*



Name space and queue id

- PPDT server context does not know ***Service ID***
  - client shall register the name string of service



Service Discovery

Fumio Nagasaka
Seiko Epson
20, Sep. 1999

Connect

- Connect specifies name of service utilizing ServiceDiscovery then gets service handle to connect

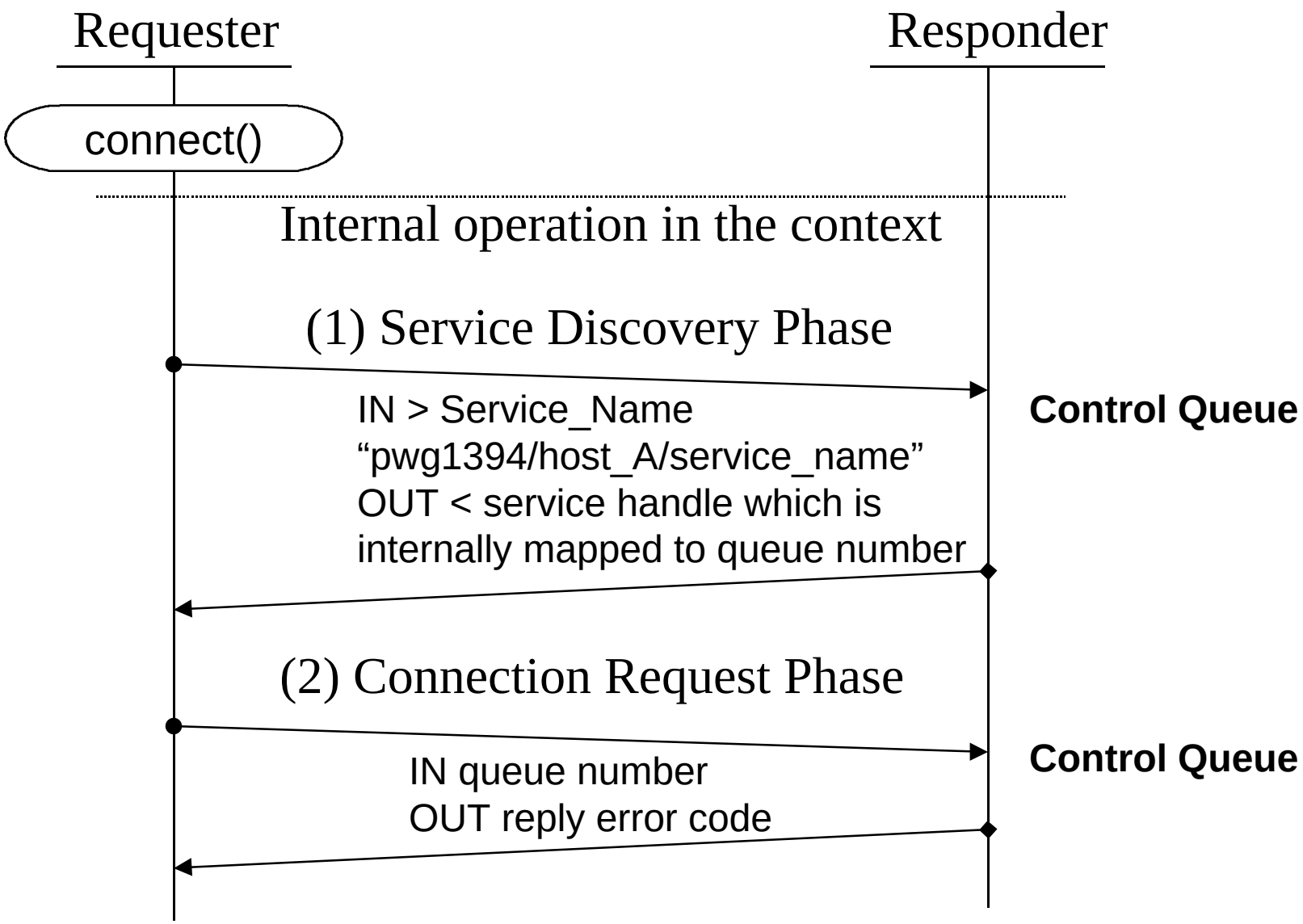
int connect(int *sockfd*, struct sockaddr **servaddr*, int *addrlen*)

sockfd : socket descriptor,

myaddr : protocol specific data structure for server's address,

addrlen : length of data structure specified in previous line,

Connect (continued …)



Send

int send(int *sockfd*, char **buff*, int *nbytes*, int *flags*);

sockfd : socket descriptor,

buff : output buffer,

nbytes : length of data in the buffer,

flags : `MSG_OOB` = send out of band data

Sendto

int sendto(int *sockfd*, char **buff*, int *nbytes*, int *flags*, struct sockaddr **to*, int *addrlen*);

sockfd : socket descriptor,
buff : output buffer,
nbytes : length of data in the buffer,
flags : `MSG_OOB` = send out of band data
to : destination address,
addrlen : length of data structure specified in previous line.

Recv

int recv(int *sockfd*, char **buff*, int *nbytes*, int *flags*);

sockfd : socket descriptor,

buff : output buffer,

nbytes : length of data in the buffer,

flags : { `MSG_OOB, MSG_PEEK` }

Recvfrom

int recvfrom(int *sockfd*, char **buff*, int *nbytes*, int *flags*, struct sockaddr **from*, int *addrlen*);

sockfd : socket descriptor,
buff : output buffer,
nbytes : length of data in the buffer,
flags : { MSG_OOB, MSG_PEEK },
from : source address,
addrlen : length of data structure specified in previous line.

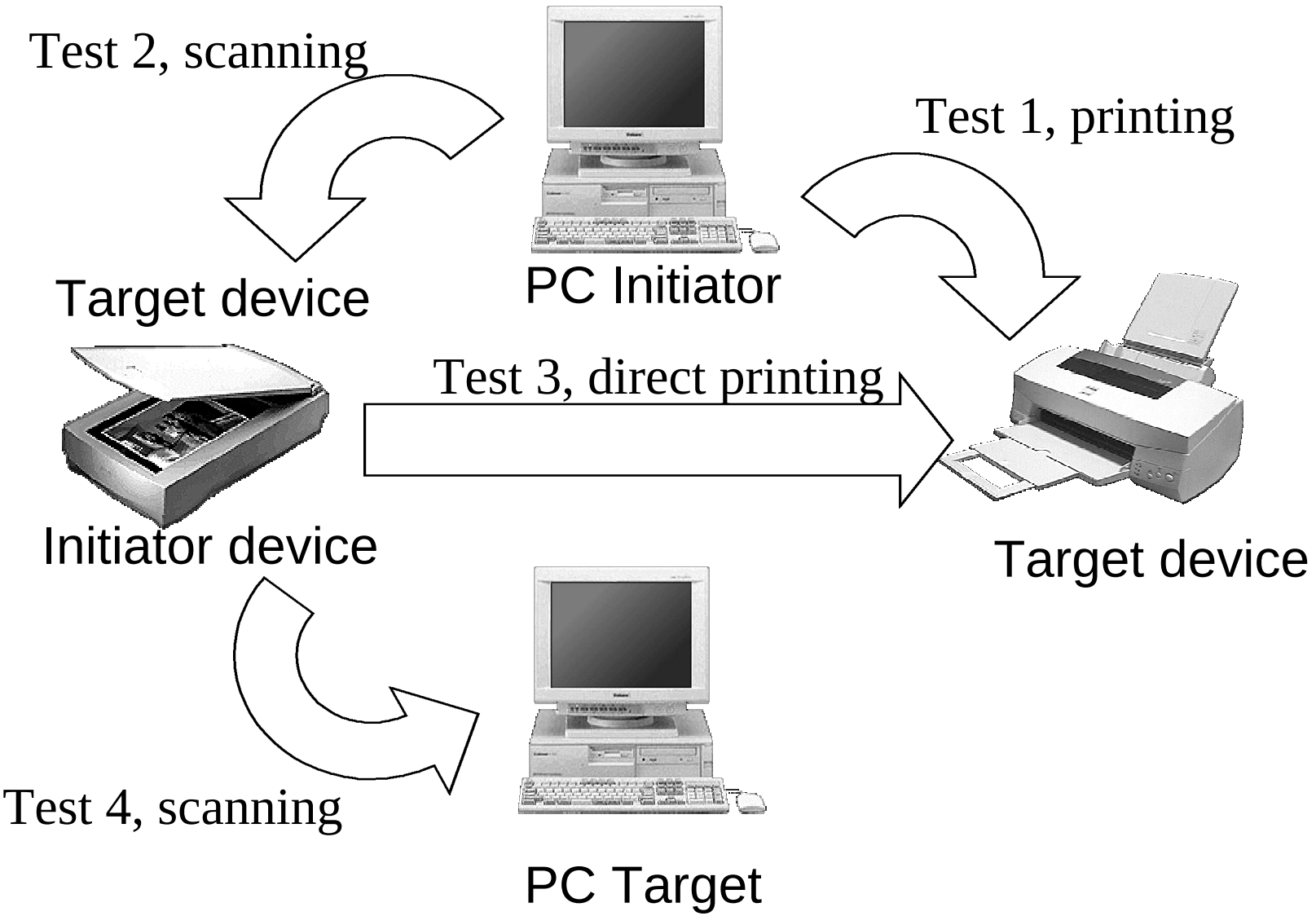
Interoperability testing for PPDT

Fumio Nagasaka

Seiko Epson

20, Sep. 1999

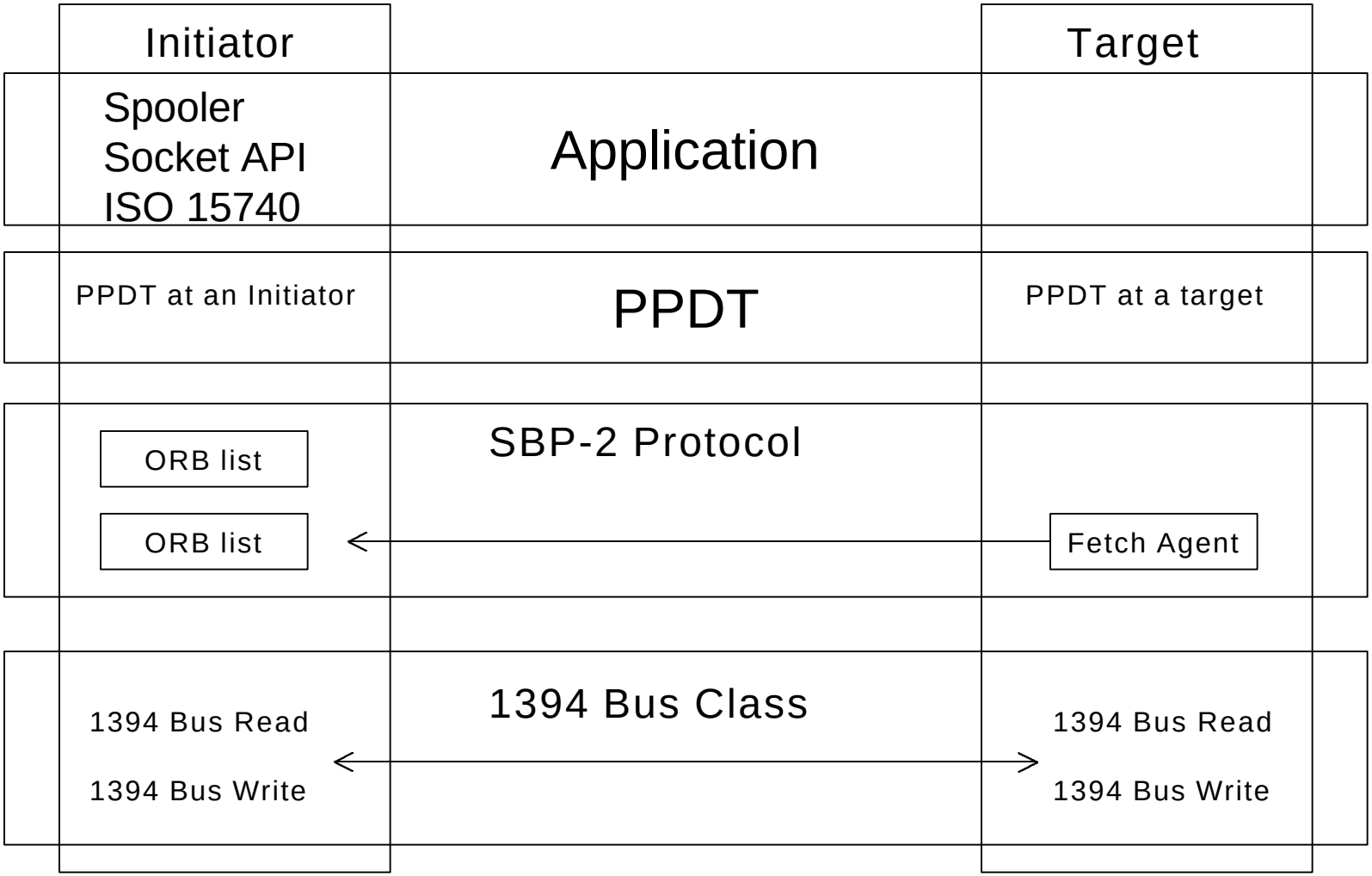
Testing cases



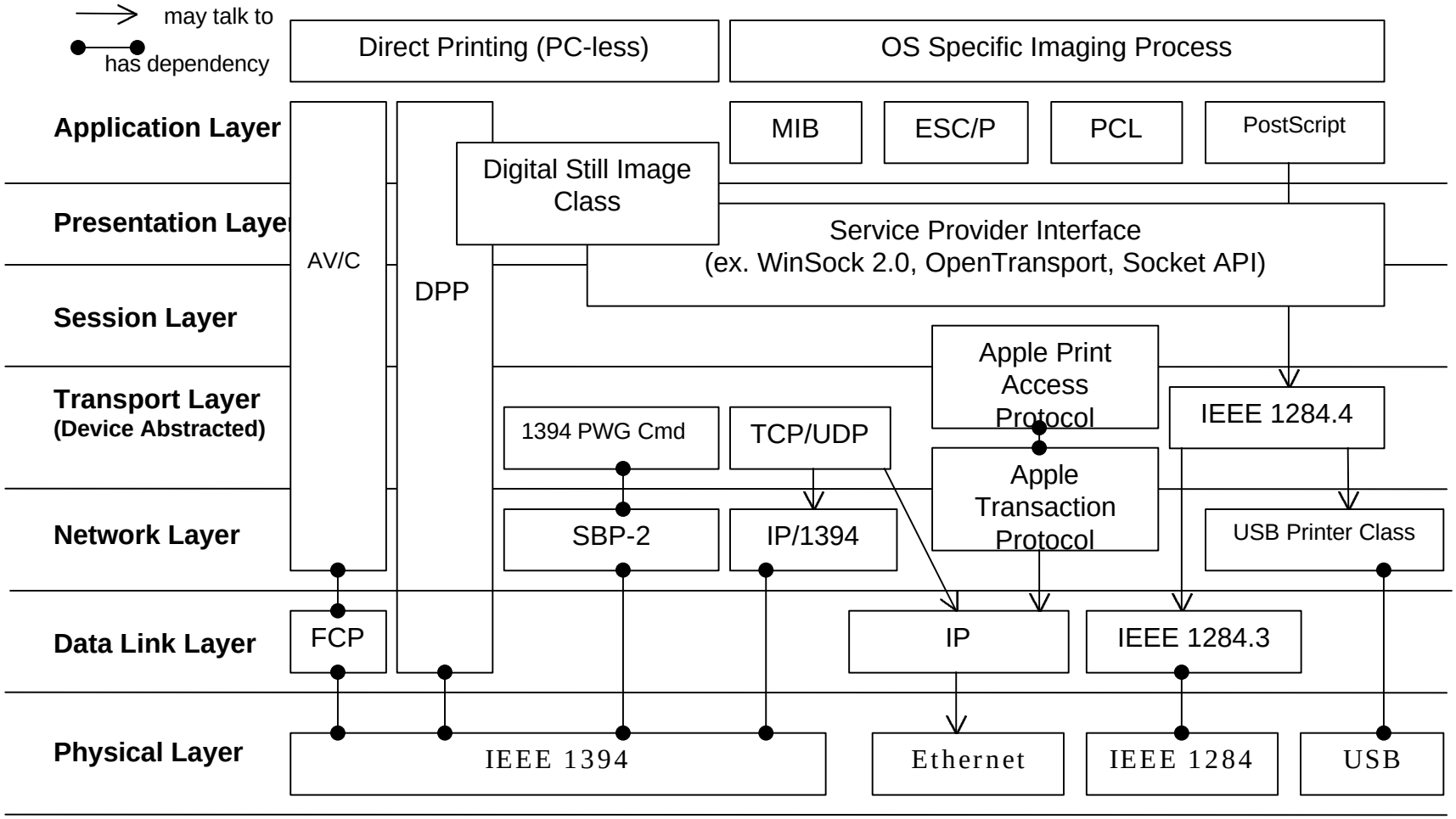
Testing cases

Case	operation	how to test
Testing 1	Printing from PC SBP-2 initiator to SBP-2 target printer	Use universal Printer Driver and PC based PPDT driver made by IHV
Testing 2	Scanning from SBP-2 target scanner to PC SBP-2 initiator	Use WIA or TWAIN application and PC based PPDT driver made by IHV
Testing 3	Direct Printing from SBP-2 initiator Scanner to SBP-2 target printer	IHV dependent solution No testing guidelines
Testing 4	Scanning from SBP-2 initiator Scanner to PC SBP-2 target	Use authorized PC PPDT server process and WIA or TWAIN compliant Scanner

Application for PPDT

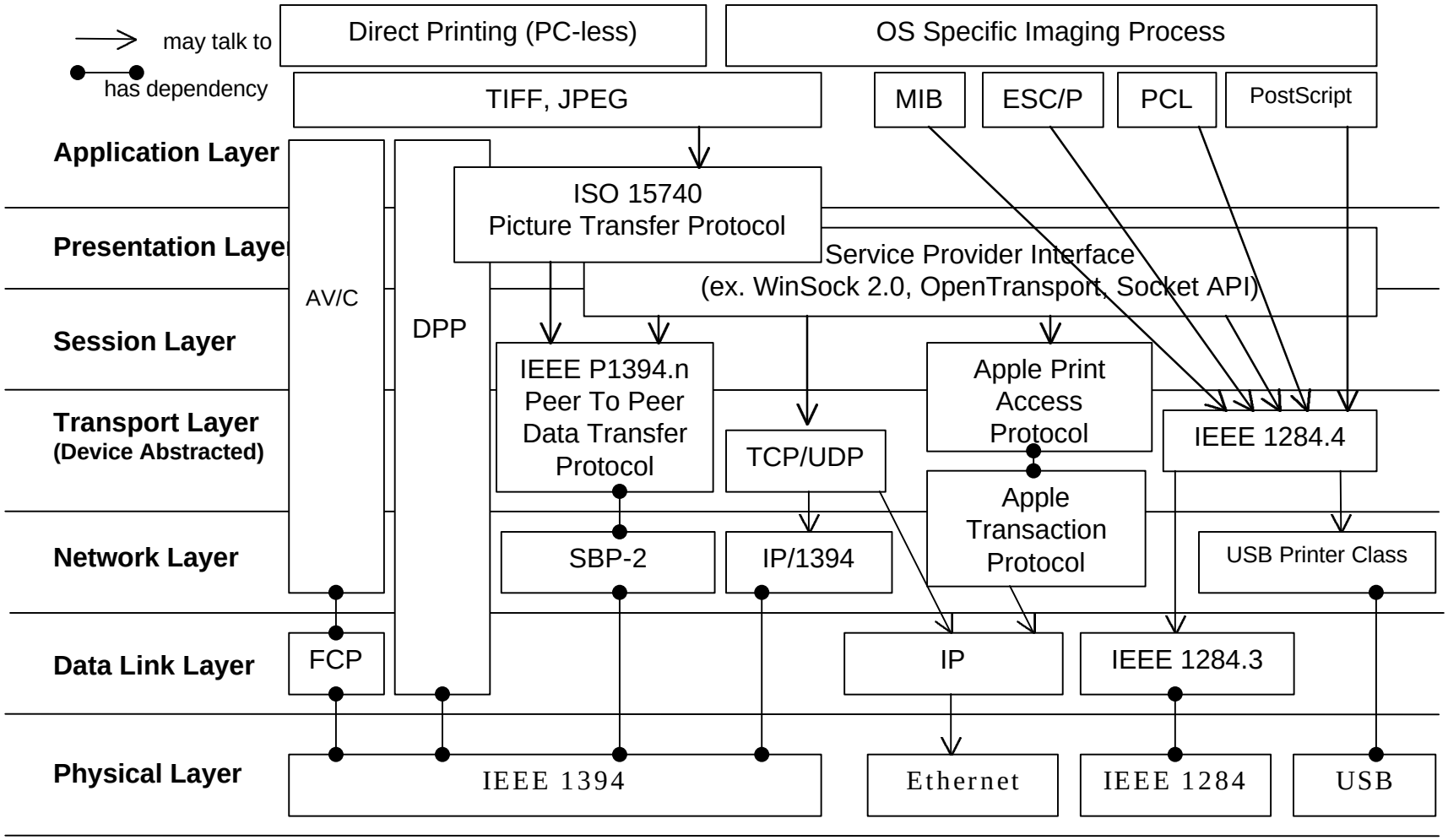


Protocol Layer (idea in 1997)



Fumio Nagasaka @ PWG meeting September 16, 1997

Protocol Layer (idea in 1999)



Fumio Nagasaka @ PWG meeting September 20, 1999

Testing requirements 1

- PC Initiator
  - PC99 or PC2001 conplient hardware
    - Windows 98 Second Edition
    - Windows 2000
  - G3/G4 Power Macintosh hardware
    - Macintosh OS 8.5 or later
  - PPDT context provided for each OS

Testing requirements 2

- PC target
  - downloadable Configuration ROM space applicable PCI 1394 board
  - IEEE 1212r compliant Configuration ROM, CSR space
  - PWG authorized PPDT server process

Testing requirements 3

- 1394 PWG standard testing application to test basic facilities of PPDT context

Thank you!

Contact me for more information

Fumio.Nagasaka@exc.epson.co.jp

EPSON Software Development Laboratory