# Open Printing Working Group Japan/Asia Activities Update

2003/06/17

Osamu MIHARA <mihara.osamu@fxpsc.co.jp>
Yasumasa TORATANI <toratani.yasumasa@canon.co.jp>

- 1. Status
- 2. Bi-di plug-in API
- 3. Vector Device Driver API
  - Idea#1 : X print service for Vector Device Support.
  - Idea#2 : API based on PS, PDF and SVG
- Schedule

# Oct,2003 meeting and Status

- Oct, 2002 Architecture Group Meeting
  - Attendees:
    - Mark Hamzy(IBM), Mihara(FUJI XEROX),
      Kido, Shimamura, Irie, Furusawa(IBM Japan),
      Kato, Nomura(EPSON), Sakashita(AXE), Yoshiyama(NEC),
      Shida, Toratani(Canon)
  - Proposals we’ve made at the meeting:
    - API between the bi-di plug-in module and the upper modules.
      - Bi-di plug-in API.
    - Generic interface between the renderer and driver.
      - Vector Printer Driver API.
- Status
  - Both groups were suspended from the beginning of 2003.
  - Mihara and I had some discussion and started again since May.

# Bi-di plug-in (1)

- Background:
  - Each printer has a different command  to readback the printer status.
  - Lack of the standard way / format to send the local printer status / capabilities to the upper system.

- Features:
  - Obtain the printer status, e.g. Ink level, Paper jam, etc. and send them to the upper modules in the standard format.
  - Obtain the printer dynamic capabilities, e.g. Stapler, Sorter, etc. and send them to the upper modules in the standard format.

- Objective:
  - Aim to be used in each printing system; CUPS, lpr, LPRng, LP, and the customized printing systems of each vendor, etc... as the common small plug-in to reduce the development time.

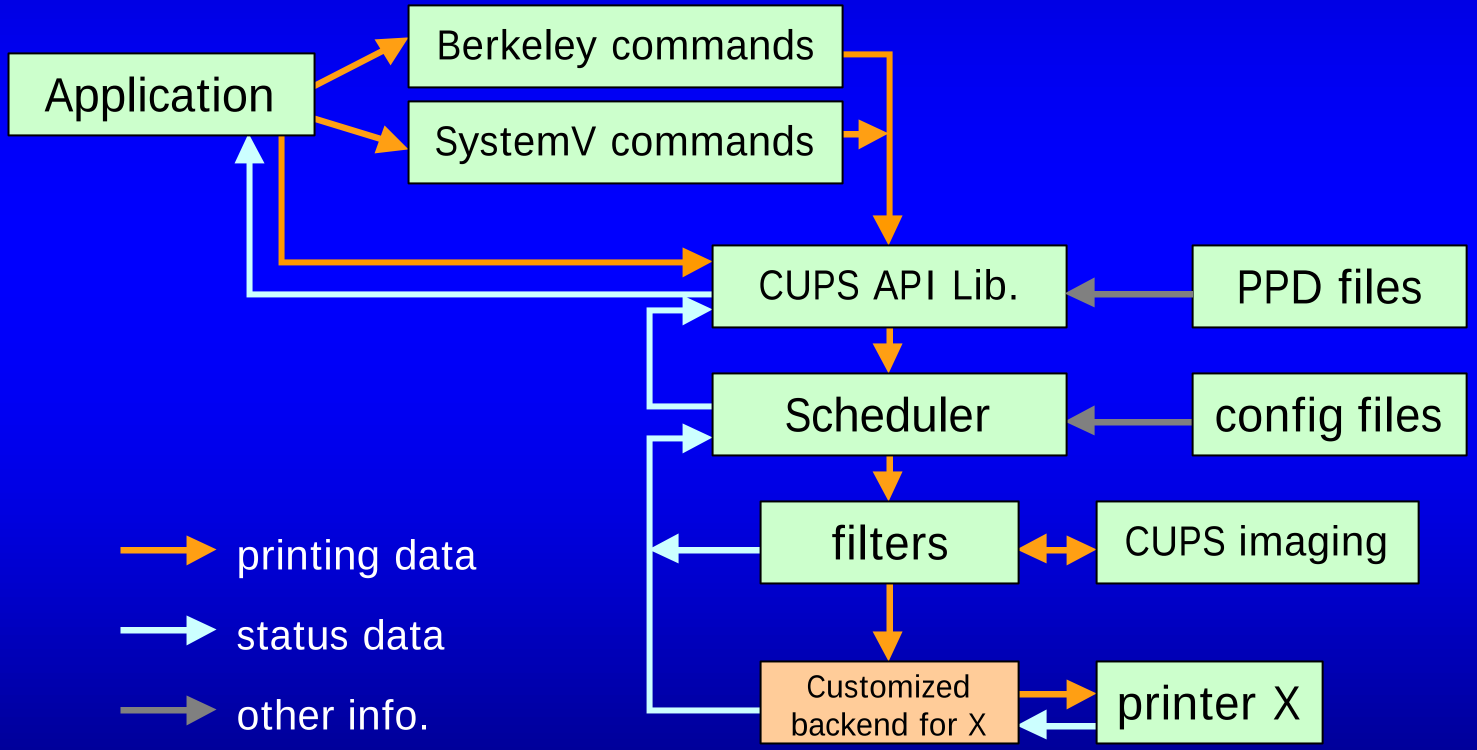
# Bi-di plug-in (2)

- The idea of the API:
  - Quite simple API for obtaining the printer info.
  - Example for the caller;

```
// Link the bi-di module or fork the bi-di process.
BidiC *pBidi = bidiNew(“bidi_module_name”, fd);
                    :
while( .... ) {
  // Obtain the printer info. and convert it to the standard format.
  bidiRead(pBidi,  pBuf, nBufBytes);
                    :
  // Send the standard format info. in the buffer to the upper system.
}                   :
// Unlink the bi-di module or kill the bi-di process.
bidiDestroy(pBidi);
```

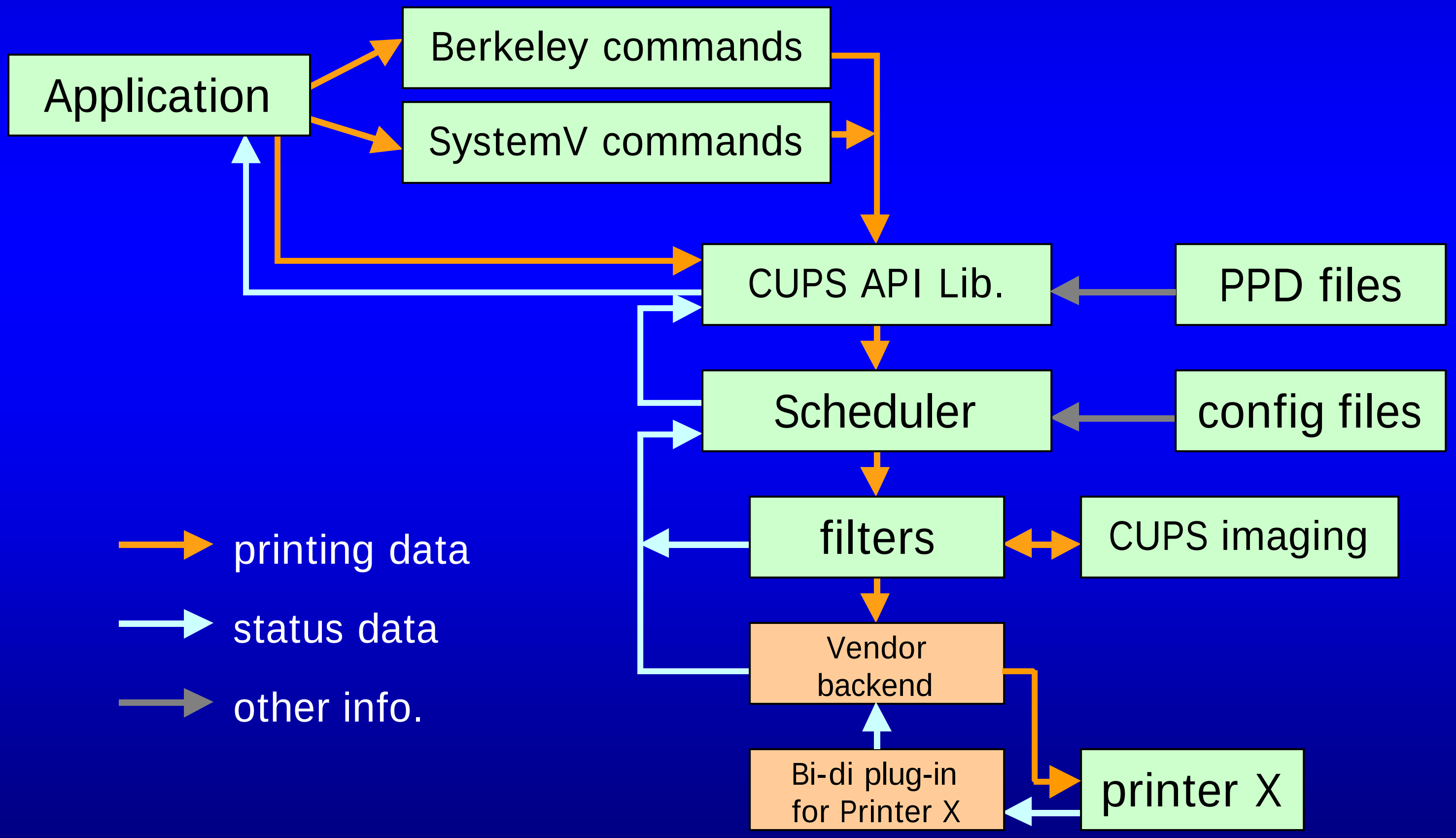
# Bi-di plug-in (3)

- CUPS 1.1.x w/o plug-in:
  - Customized backend for each device and each printer model.
    - # of backends = (# of devices; usb, parallel,etc.) x (# of printer model)



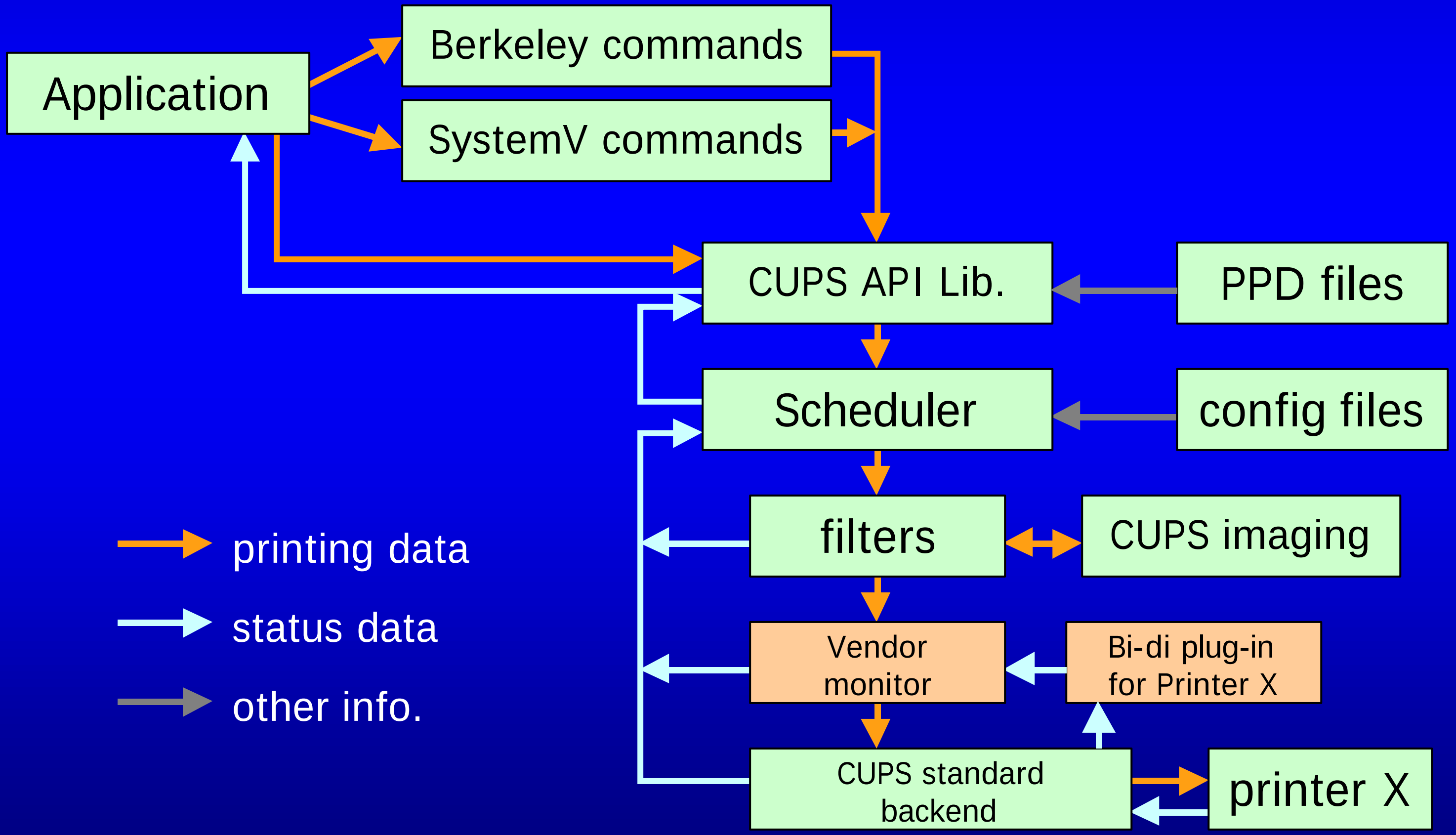
# Bi-di plug-in (4)

- Use case.1: CUPS 1.1.x w/ plug-in:
  - Vendor backend for each device and bi-di plug-in for each printer model.
    - # of vendor backends = # of devices; usb, parallel,etc.



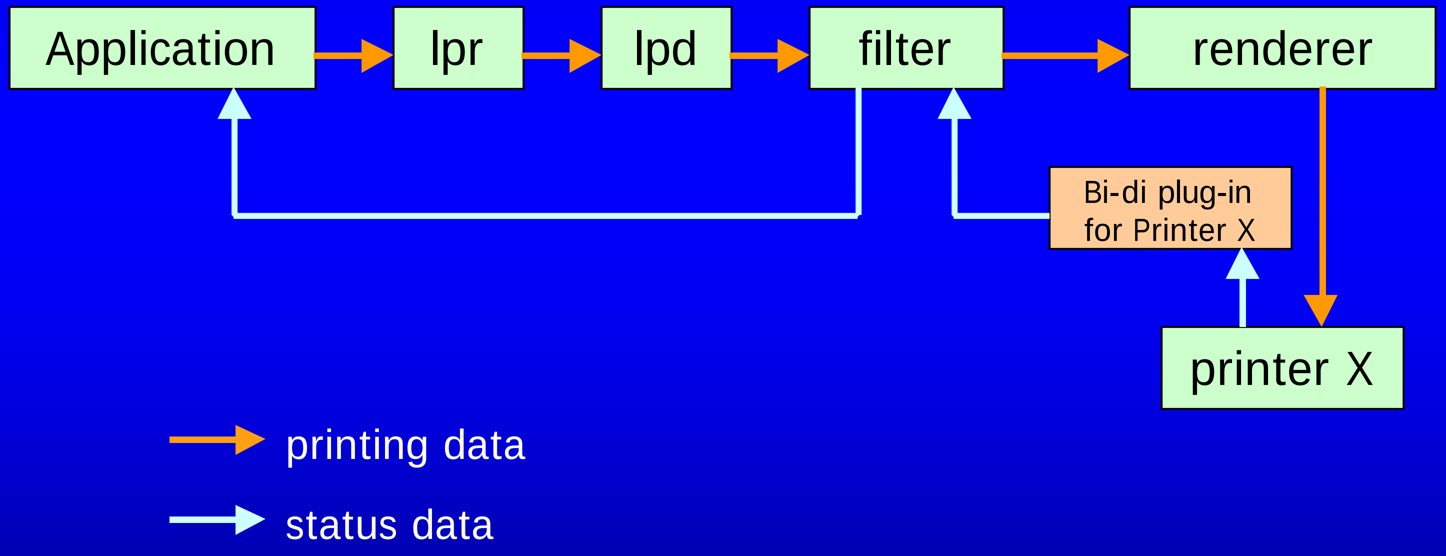
# Bi-di plug-in (5)

- Use case.2:  CUPS 1.2? w/ plug-in:
  - CUPS standard backends.
  - Vendor monitor and bi-di plug-in for each printer model.



# Bi-di plug-in (6)

- Use case.3:  Other traditional printing system:



# Bi-di plug-in (7)

- Issue:
  - Standard format of the printer status/info. sending back to the upper system.
    - XML based text format like other standards?
    - Key-Value strings pares separated by space or “,”?
    - Format adapted to IPP?
  - Standard back channel from the bi-di plug-in to the uppser system.
    - stderr ?
    - named pipe or socket?
    - Others?

# Bi-di plug-in (8)

- Steps:
  - Define the API of bi-di plug-in.
  - Define the standard data format.
  - Develop a prototype bi-di plug-in.

- Schedule:
  - TBD.

# Vector Printer Driver API (1)

- Background:
  - Lack of performance under the bitmap based driver framework.
    - Lack of use of acceleration in printer controller
    - Large data size
    - Lack of color optimization based on graphic primitives
  - Lack of support for non-PS, non-PCL PDL printers.
    - Need for the drivers for the high performance generic Vector printers.
  - Firmly linked with the renderer.
  - Depend on each renderer.
    - Need for the dynamic loading mechanism for the Vector printers.
    - Need for the renderer independent API.
- Features:
  - Generic API for UNIX/Linux.
  - Generic API for each PDL; PS, PDF and SVG.
  - Generic API for each renderer for each PDL above.
  - Generic API for shared library driver as well as IPC driver.

# Vector Printer Driver API (2)

- Issue:
  - Vector Printer Driver API is…
    - ➢ API between the **renderer** and each **vector printer driver**.
  - Renderer depends on the metafile format that the applications generate, since the renderer deals with the operators in the metafile.

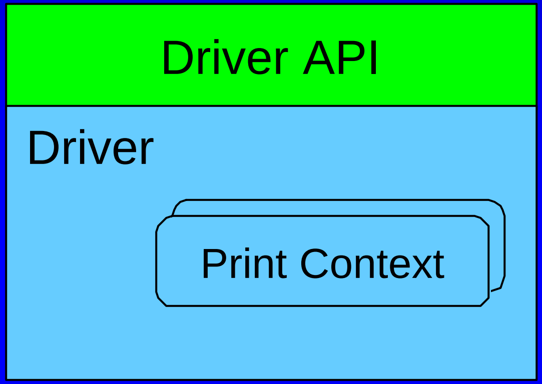
  - Which metafile format will we use mainly as the standard under Linux/Unix in the future?

⬇

  - Keep using PS and PDF, and use SVG in the future?
  - Keep using PS and PDF, and change to the new metafile based on X with the print extension in the future?

# Vector Printer Driver API (3)

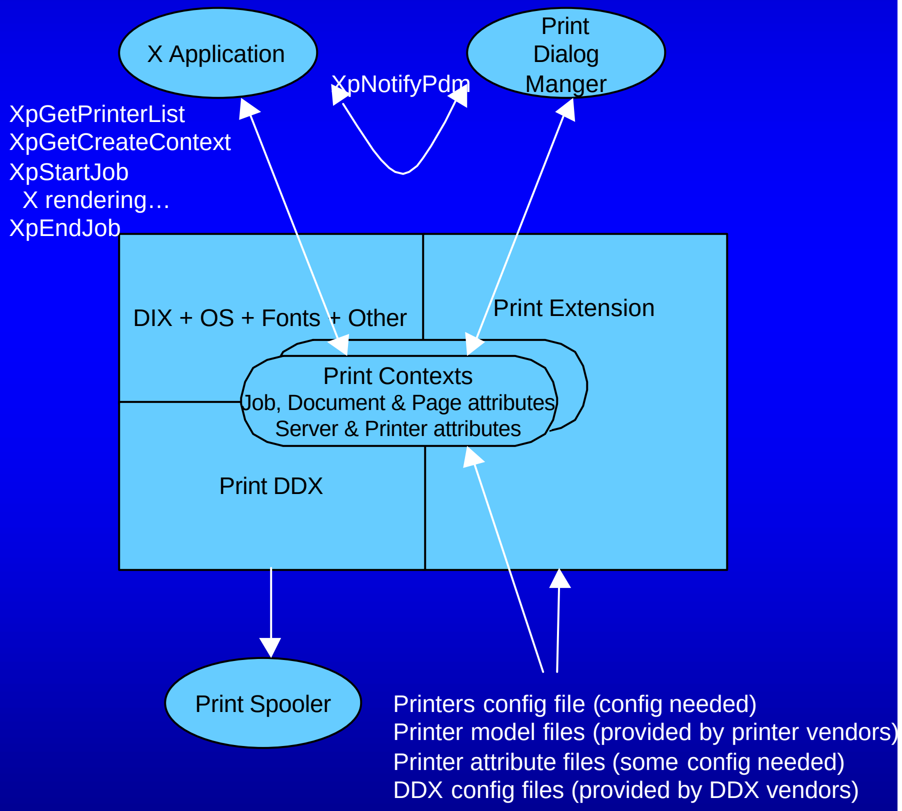
- General Structure of Vector Driver
  - Job Control Functions
    - StartDoc/EndDoc
    - StartPage/EndPage
    - Cancel
  - Graphics Context Control
    - FGcolor/BGcolor
    - Pen/Brush
    - Raster Operation (ROP)
    - Font
    - etc.
  - Graphics Drawing Functions
    - DrawPath
    - Bitmap Manupilation
    - DrawText
    - etc.
  - Pass-through data (for PS/EPS)
  - Band Control Functions
  - Job Property Control
    - Media/tray/finishing...
  - Device Configuration Management
    - Installed Options
  - etc.



# Idea #1: “X Print Service” for Vector support

- Propose “X Print Service” as vector printer driver framework and API.

- X Print Service
  - X Server with X Print Service Extension
  - Extension: operation on Context, Job, Page, Attributes, …
  - Graphics: Same service as regular X server
  - Print DDX for PCL (mono/color), PostScript and Raster are included in XC distribution
  - OpenOffice & Mozilla Supports X Print. xprint.mozdev.org



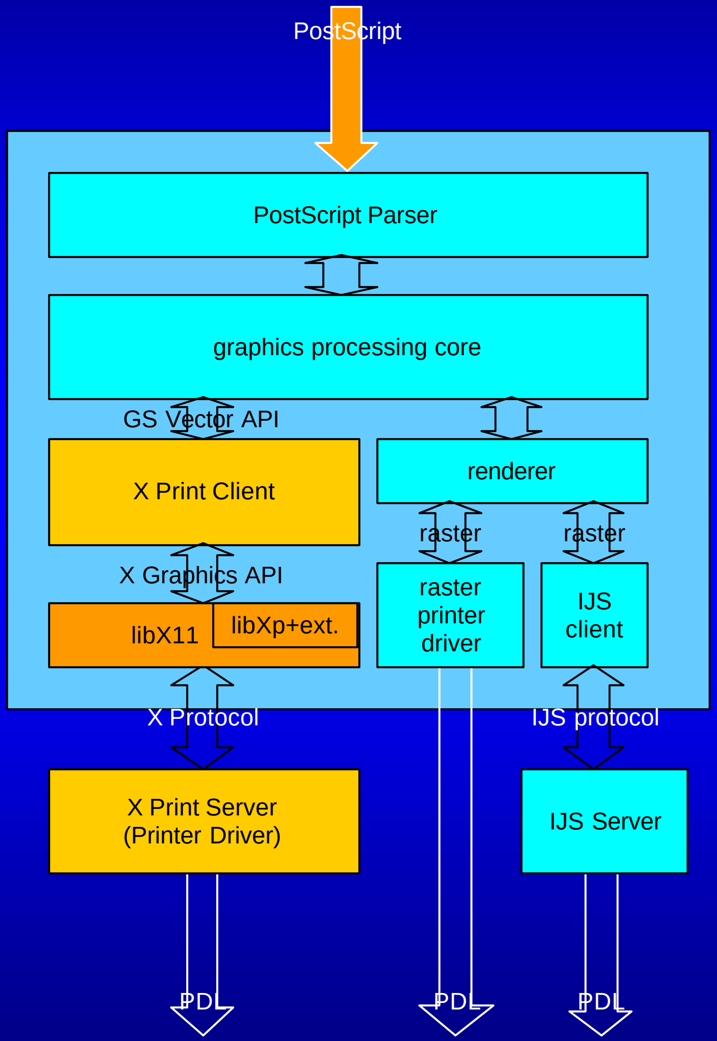
X Application
Print Dialog Manger
XpNotifyPdm
XpGetPrinterList
XpGetCreateContext
XpStartJob
X rendering…
XpEndJob
DIX + OS + Fonts + Other
Print Extension
Print Contexts
Job, Document & Page attributes
Server & Printer attributes
Print DDX
Print Spooler
Printers config file (config needed)
Printer model files (provided by printer vendors)
Printer attribute files (some config needed)
DDX config files (provided by DDX vendors)

# "X Print Service" as a Vector Driver Framework

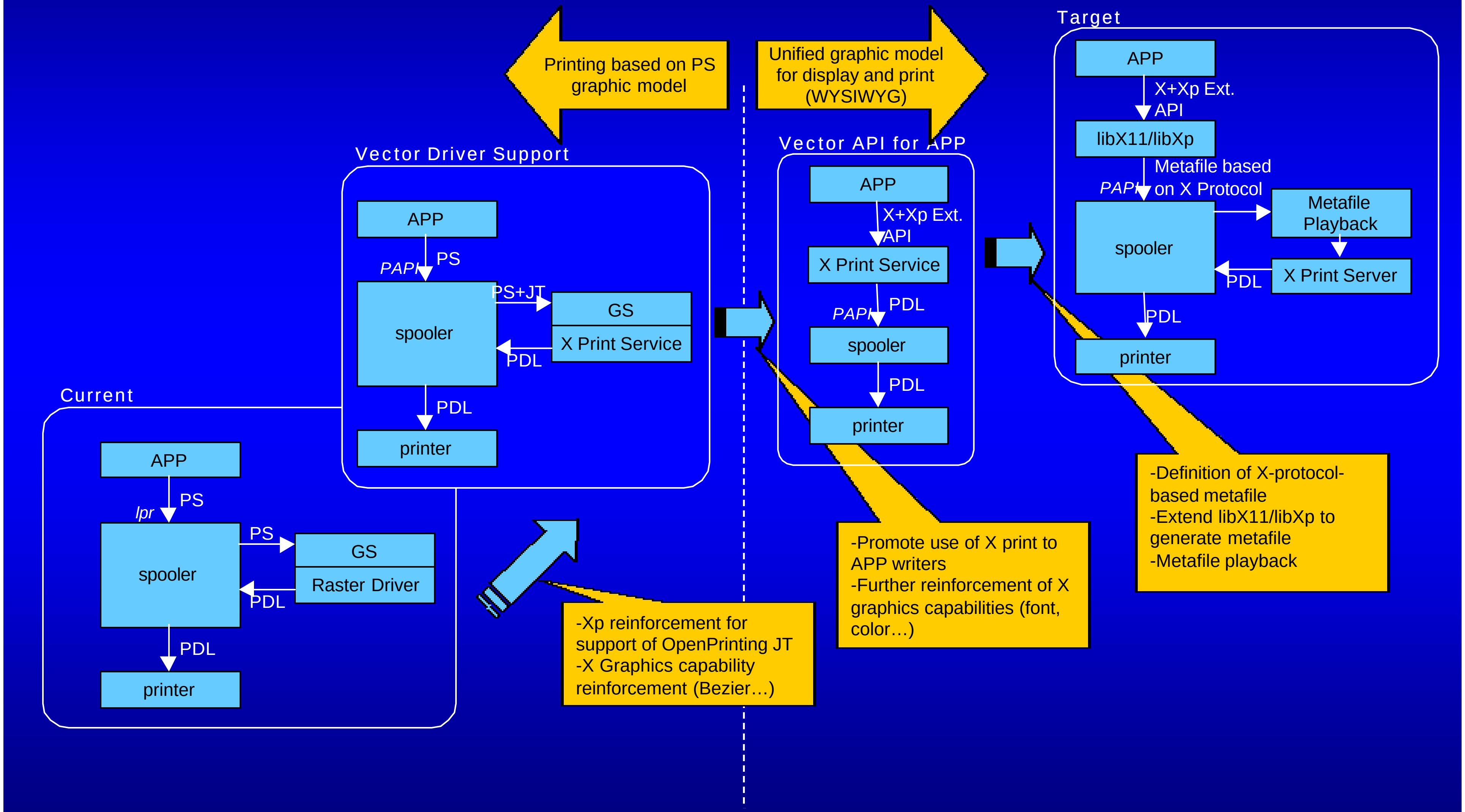
- Bunch of Merits
  - Good affinity with X applications.
    - X graphic model is natural in PC Unix world - X Tool kits (Gnome and KDE…) uses Xlib for drawing after all. X print enables same interface for printing
    - Application does not need worry about metafiles format.
  - No need to reinvent the wheel.
    - Past achievement as Graphic API set of X11 interface. - We can reuse know-how and efforts
    - Some sample implementations already exist (PCL, PostScript, Raster, PDF, etc.)
    - Ongoing project (http://xprint.mozdev.org/) … Mozilla & OpenOffice support X print service.
  - No license woes
    - Xlib (MIT license) – no problem on linking with GPL rendering programs such as Ghostscript
    - Interface between client and server is RPC based on TCP/IP
    - X Server is MIT license – printer vendor can distribute their own printer driver (DDX) in binary.
- Demerits we have to conquer
  - Old design as graphics API. Need extension for strong graphic capabilities
    - Bezier curve, raster operation, color matching
  - Only supports 16 bits (i.e. short type) coordinates system – cannot print on A0 size or banner paper in high resolution.
- Need to verify…
  - Performance – assured for video control, but need to verify with printers (higher resolution than video)
  - Required resource – code size, memory (for embedded systems and PDA's)

# First Step: Vector Support on Ghostscript using X Print

- How…
  - Built-in X print client as a vector driver into Ghostscript
  - The client translates GS vector operation into X operations.
  - The client communicate with X Print Server to generate PDL.
- So what?
  - Support APPs which generate PostScript for printing
  - No visibility change for APPs.
  - Faster printing on high end laser printers.
- Extension
  - Job Attribute via Job Ticket



# Strategy for higher level graphics printer driver support



-Xp reinforcement for support of OpenPrinting JT
-X Graphics capability reinforcement (Bezier…)

-Promote use of X print to APP writers
-Further reinforcement of X graphics capabilities (font, color…)

-Definition of X-protocol-based metafile
-Extend libX11/libXp to generate metafile
-Metafile playback

# To-do's for 1st Step

- Clarification of requirement and spec.
  - Study current Xp spec.
  - Extension to take in OpenPrint spec. such as JTAPI.
  - Extension to X Server Graphics capability
  - Dynamic configuration
    - Dynamic loading of X Print DDX
    - Device Configuration
    - Device Status
    - Interaction with User Interface settings
  - Coordination in OpenPrinting Architecture
    - Legacy AP support
    - Data flow
    - Metafile?
- Implementation
  - Client
    - Integrate Xp Client into Ghostscript
  - Server
    - Extension of Xp protocol
    - Dynamic Configuration Management
    - Spooler interface
- Verification
  - Performance
  - Application
  - Usability
- Standardization and Cooperation
  - FSG OpenPrinting WG
  - X Consortium?
  - X Print Project? (mozdev.org)
  - XFree86?

# Idea #2: API based on PS, PDF and SVG

- Graphics Model's viewpoint:
  - PS, PDF and SVG have the similar Graphics Model.
    - Path:  moveto, lineto, curveto, closepath, etc...
    - Painting:  fill, stroke, etc...
  - Graphics model of PS, PDF and SVG are wider than that of the original X.
    - Wider model can support the restricted model.
    - Restricted model can't support the wider model.
- Renderer's viewpoint:
  - Major PS renderer, e.g. Ghostscript, has the function entries of each operator to register each vector device's functions.
    - beginpath, moveto, lineto, curveto, closepath, endpath. etc...
  - SVG renderer will have the similar function entries.

# API based on PS, PDF and SVG (2)

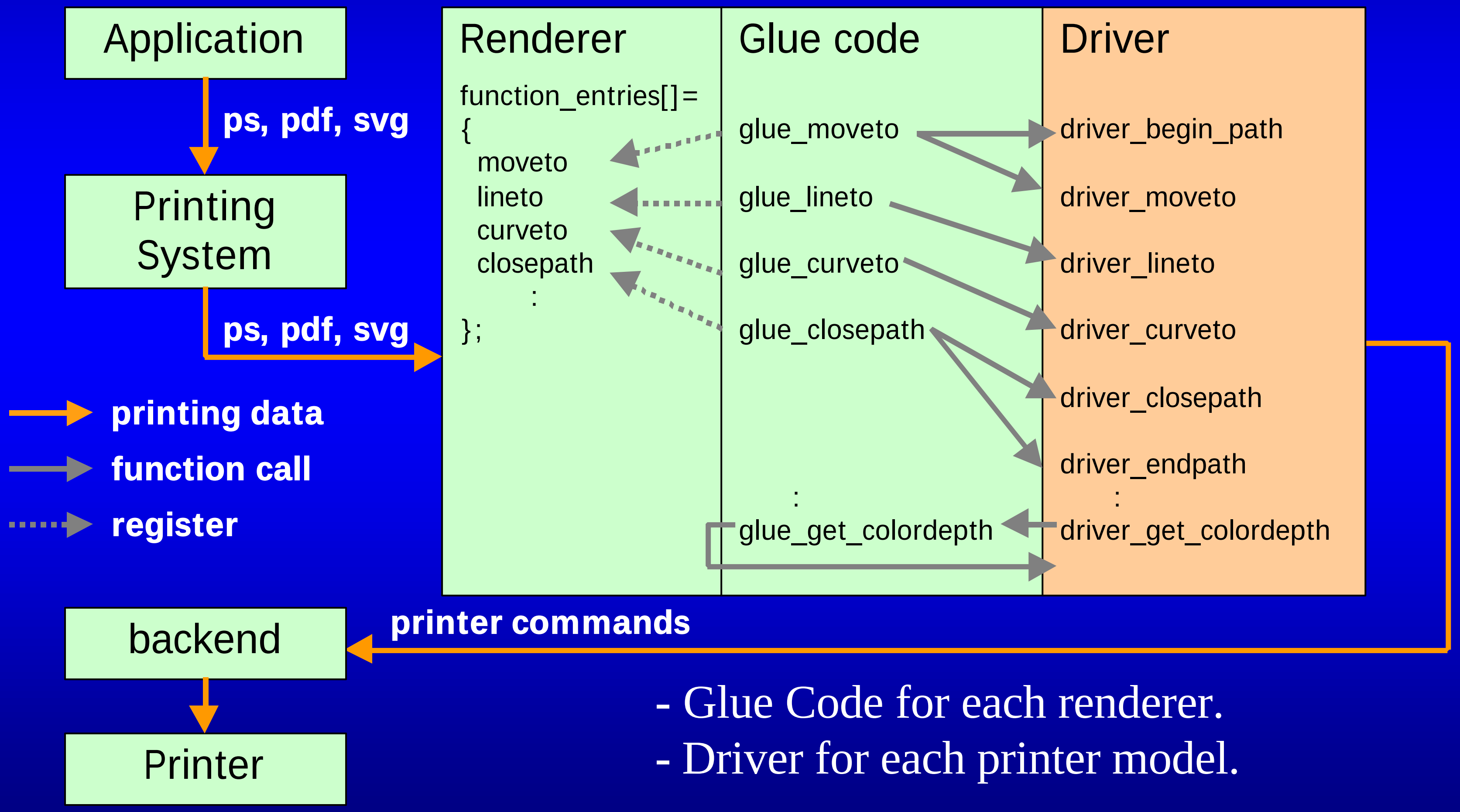
- API design policy:
  - Prepare the common function entries called from the renderer.
    - newpath, moveto, lineto, curveto, closepath, endpath, etc. (TBD)
    - setlinewidth, setcolor, setjoin, setcap, setmiterlimit, etc. (TBD)
  - Glue code linked to the renderer converts the renderer’s request to call the appropriate APIs of the driver.
  - Prepare the common callback entries from the driver to the renderer.
    - Driver can call the renderer’s function.
    - Glue code prepare the callback entries.
  - Hide the renderer dependent stuff to keep the driver generic.
    - Pass the renderer’s stuff to the driver as the generic context including the pointer to the renderer’s stuff.
    - Driver obtains several properties in the renderer’s stuff by using the common callback entries.

# API based on PS, PDF and SVG (3)

- API design policy (cont):
  - Not restrict the printer’s features.
    - If some printer support the command of “curveto”, API will support its function.

# API based on PS, PDF and SVG (4)

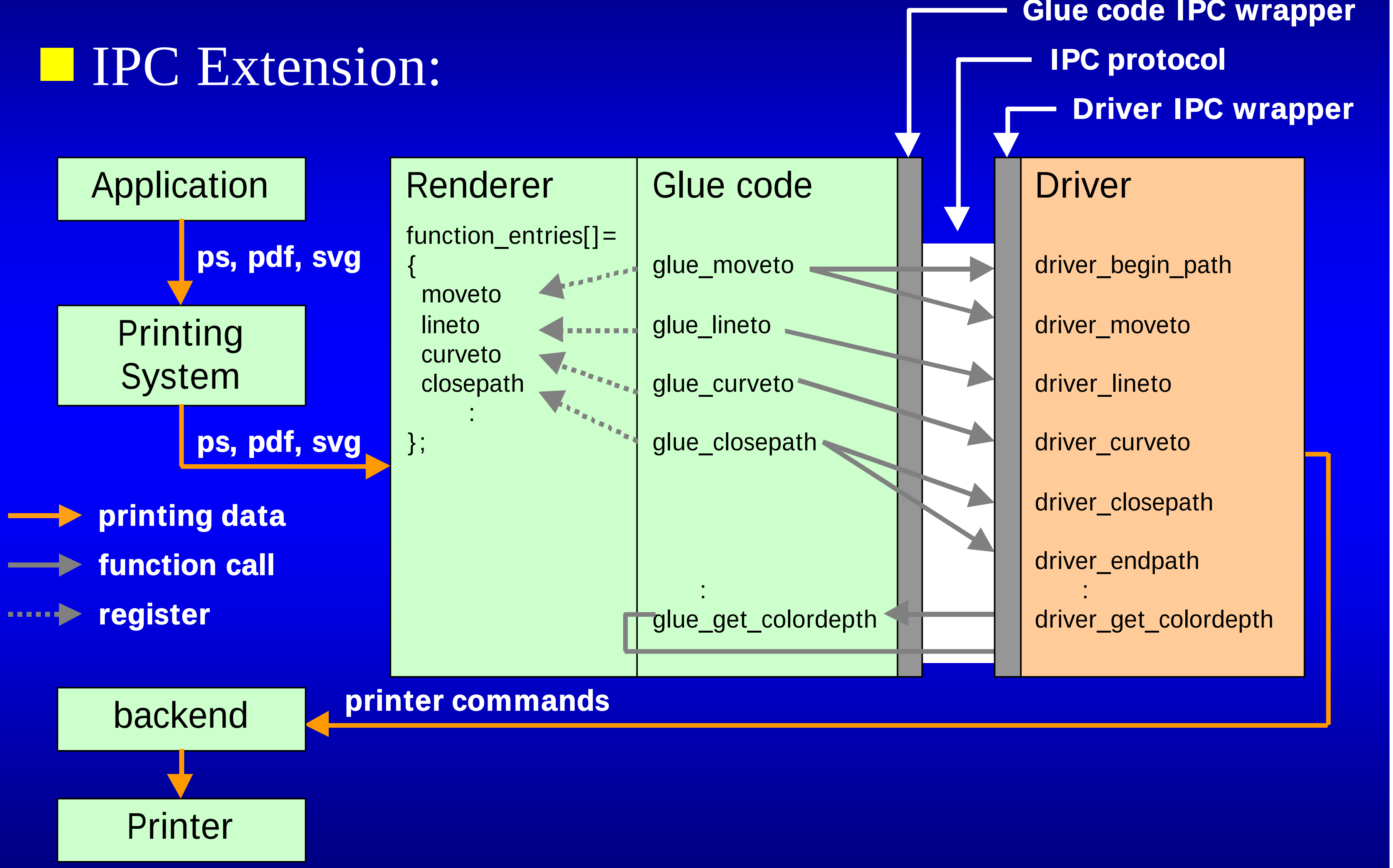
- Basic Diagram: Renderer, Glue code and Driver.



- Glue Code for each renderer.
- Driver for each printer model.

# API based on PS, PDF and SVG (5)

- IPC Extension:



# API based on PS, PDF and SVG (6)

- Steps:
  - Make a list of the function entries that renderer needs.
    - 1st implementation is Ghostscript.
  - Make a list of the function entries that printers need.
  - Select the Xprint model or PS/PDF/SVG model.
  - Define the API.
  - Define the IPC protocol.
  - Develop a prototype glue code and driver.

# Schedule

| | |
|---|---|
| End of July | Select the architecture Xprint or “API based on PS,PDF and SVG” |
| End of Sep. | Define API. |
| Oct. | Trial implementation for GS. |

# Thank you for your time and interest.

This presentation data will be stored in;

ftp://ftp.pwg.org/pub/pwg/fsg/June2003_meeting_slides/

OpenPrintWGJapan030617.ppt